TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Michael Arndt

Safe and Cost-Efficient Mobile Robot Navigation in Aware Environments

RRLAB
THE ROBOTICS RESEARCH LAB

Safe and Cost-Efficient Mobile Robot Navigation in Aware Environments

Vom Fachbereich Informatik der

Technischen Universität Kaiserslautern

zur Verleihung des akademischen Grades

Doktor der Ingenieurwissenschaften (Dr.-Ing.)

genehmigte Dissertation

von

Michael Arndt

Datum der wissenschaftlichen Aussprache	12. 12. 2016
Dekan	Prof. Dr. Klaus Schneider
Berichterstatter	Prof. Dr. Karsten Berns Prof. Dr. Paul Lukowicz

D 386

Abstract

When designing autonomous mobile robotic systems, there usually is a trade-off between the three opposing goals of safety, low-cost and performance. If one of these design goals is approached further, it usually leads to a recession of one or even both of the other goals. If for example the performance of a mobile robot is increased by making use of higher vehicle speeds, then the safety of the system is usually decreased, as, under the same circumstances, faster robots are often also more dangerous robots. This decrease of safety can be mitigated by installing better sensors on the robot, which ensure the safety of the system, even at high speeds. However, this solution is accompanied by an increase of system cost.

In parallel to mobile robotics, there is a growing amount of ambient and aware technology installations in today's environments – no matter whether in private homes, offices or factory environments. Part of this technology are sensors that are suitable to assess the state of an environment. For example, motion detectors that are used to automate lighting can be used to detect the presence of people.

This work constitutes a meeting point between the two fields of robotics and aware environment research. It shows how data from aware environments can be used to approach the abovementioned goal of establishing *safe*, *performant* and additionally *low-cost* robotic systems.

Sensor data from aware technology, which is often unreliable due to its low-cost nature, is fed to *probabilistic* methods for estimating the environment's state. Together with models, these methods cope with the uncertainty and unreliability associated with the sensor data, gathered from an aware environment.

The estimated state includes positions of people in the environment and is used as an input to the local and global path planners of a mobile robot, enabling safe, cost-efficient and performant mobile robot navigation during local obstacle avoidance as well as on a global scale, when planning paths between different locations. The probabilistic algorithms enable *graceful degradation* of the whole system. Even if, in the extreme case, *all* aware technology fails, the robots will continue to operate, by sacrificing *performance* while maintaining *safety*. All the presented methods of this work have been validated using simulation experiments as well as using experiments with real hardware.

Acknowledgments

The work on this thesis has been funded by the PhD program of the Department of Computer Sciences of the University of Kaiserslautern. The support is gratefully acknowledged. A research stay of the author at the Laboratório de Automação e Robótica (LARA) at the University of Brasília, Brazil has been supported by the PROBRAL program of the German Academic Exchange Service (DAAD) and the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES).

Remarks and Copyright

Some of the figures and formulas in this thesis have originally been created for the author's own Master's thesis. Although this Master's thesis has not been officially published, references back to it are still provided, to emphasize that this material was *not* created in the scope of this PhD thesis and to satisfy the terms of the PhD examination rules (which require that no parts of earlier examination procedures are reused in the dissertation).

Some of the work for this thesis has already been published in the proceedings of peer-reviewed conferences or in peer-reviewed journals. Content in this thesis that reproduces such already published work is clearly marked with references to the original publications. Care has been taken to use mathematical signs and symbols according to ISO 80000-2:2009, whenever possible.

[1] Those that are not marked as being based on some other work.

Contents

1. Introduction **1**
- 1.1. Motivation . . . 1
- 1.2. Objectives . . . 3
- 1.3. Structure . . . 4

2. Fundamentals **7**
- 2.1. Mobile Robotics . . . 7
  - 2.1.1. Mapping . . . 7
  - 2.1.2. Localization . . . 10
  - 2.1.3. Path Planning . . . 14
- 2.2. Aware or Smart Environments, Ambient Intelligence . . . 15
  - 2.2.1. Detection and Localization of Human Beings . . . 16
  - 2.2.2. Security and Privacy . . . 18

3. Robots in Aware Environments **19**
- 3.1. Related Research Projects . . . 19
  - 3.1.1. Physically Embedded Intelligent Systems . . . 19
  - 3.1.2. The RUBICON Project – Robotic UBIquitous COgnitive Network 21
  - 3.1.3. The ROBOT-ERA Project . . . 23
  - 3.1.4. Ubiquitous Networking Robotics in Urban Settings . . . 24
- 3.2. Other Related Work . . . 25
- 3.3. Concept for Safety, Cost-Efficiency and Performance . . . 26

4. Probabilistic State Estimation **31**
- 4.1. State Estimation . . . 31
  - 4.1.1. Motivation for Probabilistic Methods . . . 31
  - 4.1.2. State Spaces . . . 32
  - 4.1.3. Model Knowledge . . . 32
  - 4.1.4. Algorithms for State Estimation . . . 33
- 4.2. Recursive Bayesian Estimation . . . 33
  - 4.2.1. Bayesian Filters . . . 34
  - 4.2.2. Discrete Bayes Filter . . . 35
  - 4.2.3. Kalman Filter . . . 35
  - 4.2.4. Particle Filter . . . 37
- 4.3. Graph-Based State Space . . . 38
  - 4.3.1. Modeling and Implementation . . . 39

4.3.2. Visualization and Mapping to a Grid . . . 40
4.4. Brownian Motion on Graphs . . . 46
4.4.1. Implementation and Examples . . . 48
4.5. Sensor Model for PIR Sensors on Wireless Sensor Nodes . . . 50
4.5.1. Calculation of the Confidence . . . 55
4.5.2. Transformation between Global and Sensor Coordinates . . . 57
4.5.3. Visualization of the Confidence Function . . . 60
4.5.4. From Confidence to Direct Sensor Model . . . 61
4.5.5. Direct Sensor Model for Multiple Sensors . . . 63
4.5.6. From Confidence to Marginalized Direct Sensor Model . . . 64
4.5.7. From Direct to Marginalized Inverse Sensor Model . . . 64
4.5.8. Marginalized Inverse Sensor Model for Multiple Sensors . . . 66
4.5.9. Conclusion and Other Sensors . . . 67
4.6. Single-Target Tracking . . . 67
4.6.1. Target Tracking using Bayesian Filters . . . 67
4.6.2. Experiments . . . 68
4.7. Multi-Target Tracking . . . 72
4.7.1. Data Association . . . 72
4.7.2. Related Work and Anonymous, Sparse Sensor Instrumentation 74
4.7.3. Multiple Hypothesis Tracker (MHT) . . . 75
4.7.4. Probability Hypothesis Density (PHD) Filter . . . 76
4.7.5. Experiments . . . 77
4.8. Occupancy Graph Mapping . . . 78
4.8.1. Relation to Target Tracking . . . 79
4.8.2. Occupancy Graph . . . 80
4.8.3. Particle Initialization . . . 81
4.8.4. Update of Particle Weights . . . 81
4.8.5. Example Update Sequence . . . 82
4.8.6. Experiments . . . 82

5. Safe and Cost-Efficient Navigation 85
5.1. Virtual Obstacle-Avoidance Sensors . . . 85
5.1.1. Relation to Previous Works . . . 86
5.1.2. Relevant Particles on the Occupancy Graph . . . 87
5.1.3. Local Sensor Systems . . . 92
5.1.4. Experiments . . . 93
5.1.5. Application to Occupancy Grids . . . 97
5.2. Risk-Reduced Path Planning . . . 98
5.2.1. Relation to Previous Works . . . 98
5.2.2. Human Aware Planning . . . 99
5.2.3. Risk and its Quantification . . . 100
5.2.4. Fusion of Risk and Distance . . . 106
5.2.5. Experiments . . . 107
5.2.6. Discussion . . . 110

5.3. Predictive Path Planning . . . 111
5.3.1. From Risk Assessment to Risk Prediction . . . 111
5.3.2. State Prediction in Robotics . . . 112
5.3.3. Prediction using Bayesian Filters . . . 113
5.3.4. Planning with Prediction . . . 113
5.3.5. Experiments . . . 116
5.3.6. Discussion . . . 117
5.4. Robot Localization in Aware Environments . . . 118
5.4.1. Relation to this Thesis . . . 119
5.4.2. Fundamentals of Radio Communication . . . 120
5.4.3. Learning the Radio Map . . . 123
5.4.4. Localization within the Radio Map . . . 124
5.4.5. Experiment . . . 125
5.5. A Second Robot . . . 128
5.5.1. Knowledge Exchange . . . 128
5.5.2. Modifications to the Sensor Model . . . 129
5.5.3. Experiment . . . 130

6. Application Study **133**
6.1. Experimental Setup . . . 133
6.2. Experimental Results . . . 134
6.2.1. Global Localization using RSS Fingerprinting . . . 134
6.2.2. Enhancement of the State Estimation by the Second Robot . . . 136
6.2.3. The Influence of Prediction . . . 137
6.3. Discussion . . . 139

7. Summary and Conclusion **141**
7.1. Summary . . . 141
7.2. Conclusion . . . 142
7.3. Future Work . . . 144

A. Description of Experimental Platforms and Components **145**
A.1. The AmICA Wireless Sensor Network Platform . . . 145
A.1.1. Overview . . . 145
A.1.2. Signal Processing . . . 146
A.1.3. Security . . . 147
A.2. The Mobile Robot Artos . . . 147
A.2.1. Overview . . . 147
A.2.2. Mechanics . . . 148
A.2.3. Hardware . . . 148
A.2.4. Sensor Systems . . . 149
A.2.5. Control System . . . 151
A.3. Offices of the Robotics Research Lab . . . 152
A.3.1. Floor Plan . . . 153

A.3.2. Graph Layout . . . 153
A.3.3. Sensor Model . . . 155
A.3.4. Motion Model . . . 157
A.4. The Simulation Environment . . . 157
A.4.1. Overview . . . 157
A.4.2. The Environments . . . 158
A.4.3. The Robot . . . 158
A.4.4. The AmICA Nodes . . . 159

B. Derivation of Formulas and Proofs **163**
B.1. Nyquist-Shannon Theorem applied to Mapping from Graph to Grid . 163
B.2. Braking Distance and Safe Velocity . . . 165
B.3. Calculation of the Value of d_ε for Virtual Obstacle Avoidance . . . 166
B.4. Monotonicity of the Safety Cost Function . . . 167
B.5. Integral of the Safety Cost Function . . . 168
B.6. Incremental Update Rule for the Arithmetic Mean . . . 168
B.7. Cartesian Equation of an Elliptic Cone . . . 169
B.8. Intersection of a Line with an Elliptic Cone . . . 171
B.9. Intersection of a Line with a Sphere . . . 172
B.10. Calculation of Transformation Matrices for the Sensor Model . . . 173
B.10.1. Forward Transformation: From Sensor To Global Coordinate . 173
B.10.2. Inverse Transformation: From Global to Sensor Coordinate . . 174

C. Additional Visualizations **177**
C.1. Single-Target Tracking . . . 177
C.2. Multi-Target Tracking and Occupancy Graph Mapping . . . 182
C.3. Virtual Obstacle-Avoidance Sensor . . . 190
C.4. Enhancement of the State Estimation by a Second Robot . . . 197

Bibliography **199**

Index **213**

1. Introduction

1.1. Motivation

Today, there is a rising number of mobile robots performing service tasks in everyday domains. In households, vacuum cleaning and scrubbing robots are well-known examples. In industry, there are autonomous transport systems, so called *automated guided vehicles* [Hägele 08, 42.1], that perform tasks such as delivering goods between places. Even in outdoor environments, mobile robots have evolved from pure research projects in isolated environments [Buehler 07, Buehler 09] to autonomous self-driving cars on public roads.

In parallel, so called *ambient* or *smart* technology finds its way into everyday life [Augusto 10]. This technology, in the form of sensors embedded into the environment, turns ordinary environments into so called *aware environments*. Examples include motion detectors embedded into office or home environments to automate the lighting. In the scope of autonomous cars, large-scale smart environments can consist of "smart" traffic infrastructure, such as inductance loop detectors or even other traffic participants (see the domain of "connected cars").

The goal of this thesis is to bring robotics and ambient technology a bit closer together in order to increase safety, cost-efficiency and performance of mobile robotics. However, before introducing the ideas and methods that are required to do so, the classic situation in mobile robotics shall first be illustrated. Figure 1.1 shows a radar chart[1] which depicts three typical, but usually opposing design goals in robotics. Namely, those are *safety*, *low-cost* and *performance*.

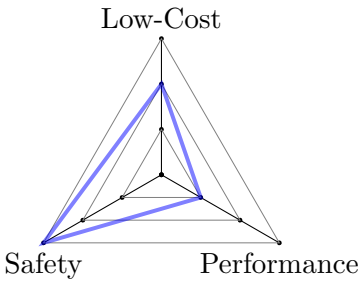


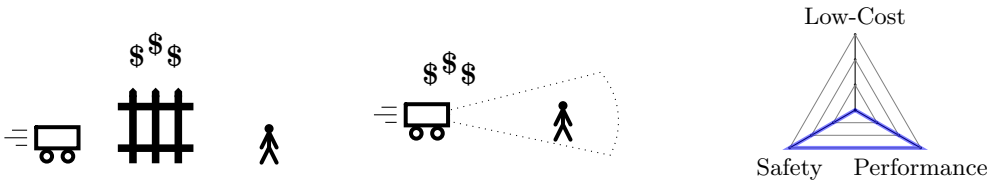
Figure 1.1.: The Safety/Low-Cost/Performance triangle.

[1] Sometimes it is also called a spider chart.

A system can be visualized in the radar chart by connecting the three values of its parameters with line segments. The result is a triangle which describes the relation of safety, low-cost and performance of the situation. In the figure, such a triangle is drawn for an example system which has low cost and very high safety, but also low performance.

For autonomous mobile robots, safety is usually one of the highest goals and thus seldom disputable. However, a high level of safety is usually "purchased" with higher system costs and/or reduced performance. Consider the following two examples: In the first example, a high-speed mobile robot (or a fleet thereof) is made highly safe by strictly separating the environment into a part for human beings and a part for robots, e.g. using a fence. This strategy allows to optimize safety as well as performance (i.e. the robots can drive with high speeds, as it is ensured that they will not encounter any human "obstacles"). On the other hand, costs are increased, because the fence needs to be installed and maintained. Additionally, the division of the environment may lead to other drawbacks, as reduced space for human workers or may not be feasible at all (like in home or office environments). This strategy is therefore usually only used in industrial environments.

For the second example, consider the same initial situation. But now, instead of installing a fence, the robots are outfitted with safety-certified high-precision laser rangefinders that are able to stop the robots in every critical situation involving a human being. Again, the same situation as before arises – safety and performance are both high, but the cost of the system is also high due to the complex sensors. See figure 1.2 for a visualization of these situations.



(a) Fast robot, separation of environment.

(b) Fast robot, expensive local sensors.

(c) Visualization of the situations in the triangle.

Figure 1.2.: Depiction of two situations with high safety, high performance, but high costs.

So what else could be done to optimize the system with respect to costs and performance without sacrificing safety? Starting again with the same initial situation, it is possible to reduce the speed of the robots, so that even with only simple (i.e. cheap) sensors, they are able to sense people early enough to avoid collisions and injuries. This strategy definitely keeps the costs low while providing a high level of safety. However the performance of the robots clearly decreases as lower speeds lead

to longer times to complete tasks and thus reduced productivity, see figure 1.3 for a visualization of this situation.

(a) Slow robot, no separation of environment.

(b) Visualization of the situation in the triangle.

Figure 1.3.: High safety, low-cost, but poor performance.

1.2. Objectives

The primary objective of this thesis is to simultaneously maximize *all* of the three parameters safety, low-cost and performance. This goal is achieved by – figuratively speaking – shifting the complexity away from the local sensor systems of the robots into external sensors distributed in the environment and suitable processing of the data gathered through theses sensors.

The core of this thesis consists of two parts that are closely related to each other. The first part is the *estimation of the state* of the aware environment using mainly sensors that are embedded into the environment itself. This thesis makes use of wireless sensor nodes that are equipped with passive infrared sensors. Those are capable of detecting motion (i.e. people). The raw sensor events are processed using algorithms that are based on the principle of *recursive Bayesian estimation*. These algorithms make use of *models* for the evolution of the state (motion models) as well as for the characteristics of the sensors (sensor models) to estimate the state according to some definition of the *state space*. Chapter 4 discusses the probabilistic algorithms that are used throughout this work, as well as the definition of the state space (which is based on a graph structure) and the implementation of the models for this work.

The other core part is centered around applications in mobile robotics that mainly make use of the results of the state estimation process. These are e.g. methods that extend the range of the robots' sensors for obstacle avoidance, so that the robots are able to drive with a higher speed than their local sensor systems permit. Another application is the use of the estimation results during path planning of mobile robots. This allows to plan paths that are not only *short* (i.e. fast), but also *safe* with respect to the distance between the robots and humans in the environment. These applications are primarily described in chapter 5. The thesis also shows the possibility of the other

direction of interaction between those two parts: Data from the local sensors of a mobile robot are shared in the environment and can in turn be used to improve the state estimation process.

Another important objective is to reach all these goals under certain *constraints*. It should be possible to make use of *existing infrastructure* whenever possible and the sensor instrumentation of the environment should be *sparse*. This means that already small installations of aware technology should lead to an improvement of the three parameters of safety, low-cost and performance. The sensors itself should be *passive* in a sense that no explicit interaction of the people is required for the system to work. This specifically prohibits any use of active tags or devices that the people need to carry. Also, only sensors that do not extensively invade the *privacy* of people should be used (i.e. no cameras or other sensors that deliver rich data). Finally, another goal is to be *fault-tolerant*. Sensor failures should be easily detectable and even under such circumstances, the system as a whole must not stop working.

1.3. Structure

The rest of this thesis is structured as follows: The goal of chapter 2 is to give an overview of the fundamentals of this work. This includes fundamentals from the domains of mobile robotics as well as from the domain of aware environments. This chapter may be skipped by the reader if he or she feels comfortable with these foundations. However, it does introduce and discuss important aspects (including definitions) which are resumed later throughout the thesis.

A total of four larger and relevant research projects that are related to the work of this thesis are first presented in chapter 3. After these, some smaller, but still relevant works are briefly introduced. At the end of that chapter, based on the findings in the literature, the central theses of this work are developed. These are centered around safety, cost-efficiency and performance of mobile robots in aware environments.

Chapter 4 introduces probabilistic methods for state estimation, based on recursive Bayesian estimation. Several typical algorithms that are commonly used for that purpose are discussed. It also describes the design of the state space as well as the models for motion of people as well as for the ambient sensors used in this thesis (passive infrared sensors on wireless sensor nodes). The chapter also presents three applications of recursive Bayesian estimation: Single and multi-target tracking as well as the so called occupancy graph mapping. Also, first experimental results are presented there.

The following chapter 5 builds upon these probabilistic methods to create applications in robotics according to the previously introduced theses and goals. The first application that is introduced there makes use of the data to improve local obstacle avoidance of a mobile robot. The following two applications handle global path planning, to be able to find safe, yet cost-efficient paths through the environment. While the first global planner only assesses the current situation as estimated by the probabilistic

methods, the second one uses prediction to analyze the evolution of the environment, originating from the current situation. The idea to assist the global localization of a mobile robot using received signal strength measurements of wireless transmitters in the aware environment is also introduced in that chapter. Additionally, an example on how to extend the state estimation process using other sensors (in form of a second robot) is given.

The previously summarized chapters do show experimental evaluations of the proposed methods. However, they all regard their concepts and results independently from each other. In contrast, chapter 6 is dedicated to an application study which demonstrates the system *as a whole*, making use of the methodologies proposed and demonstrated in the earlier chapters. The final chapter 7 concludes the thesis and presents an outlook on possible future work.

The appendix is divided into three parts. The first one, appendix A describes the experimental platforms used in this work in detail. This includes the wireless sensor network platform, the mobile robots and the aware environment as well as the simulation framework used to perform simulation experiments. The second one, appendix B is a collection of derivations of formals and proofs that are needed for the main chapters, but were considered to be too verbose to be replicated there. The last part of the appendix, appendix C solely contains further visualizations (mainly of experiments) that, due to space constraints, do not fit into the main chapters of this thesis.

The appendix is followed by the bibliography with back-references to the pages in which they have been cited. The following index contains important keywords and references to their occurrences in the text.

2. Fundamentals

This chapter aims to provide an overview about fundamental concepts and ideas of the two main domains with which this thesis is concerned: Mobile robotics and aware environments. The goal is to establish enough foundations for a reader with either no background in robotics or aware environments (or none of both) to be able to understand the ideas of this thesis.

The concepts and foundations contained in this chapter are all relevant to one or more sections in the later chapters of this thesis. For this reason, they are bundled here. The whole chapter or parts if it may be skipped if the reader feels familiar with the content or decides to look up the information only when needed. To support this non-linear reading order, the later sections provide references back to the corresponding fundamentals.

2.1. Mobile Robotics

Mobile robots are robots capable of moving through their target environment by themselves – in contrast to e.g. stationary industrial robots. How this locomotion is achieved does not matter for the definition. On the ground, wheels [Campion 08] or legs [Kajita 08] are two possible means of providing locomotion, but more exotic possibilities exist, e.g. snake-like locomotion [Murphy 08, 50.3.5]. Besides robots that move on the ground, there are also mobile robots that swim [Meyer 08, 60.4.1], dive [Antonelli 08] and of course ones that fly [Feron 08].

Most of the ideas of this thesis are mainly applicable to mobile robots, and they cover several important aspects of mobile robotics. This section aims to introduce the aspects that are revisited or needed in later chapters. It starts with a short introduction of mapping that is highly connected to the process of localization which follows. The section is concluded with path planning of mobile robots.

2.1.1. Mapping

Mapping in mobile robotics describes the process of creating a representation of the environment in form of a map. The *exact* type of this representation usually depends on the application. Often, the map is used to classify the environment into occupied and free space. Usually, free space is *traversable* by the robot, whereas occupied space denotes *obstacles* that cannot be traversed and must be avoided. Occupancy grid

maps are a popular probabilistic method of achieving this classification. They work by dividing the environment into single cells arranged on a grid-like structure [Elfes 89]. A different way to present the environment is using structures based on graphs[1], e.g. in the form of topological maps [Burgard 08, 36.2.3]. Both the occupancy grid as well as the graph-structure are important for this thesis and are used in the later sections, especially when handling state spaces (section 4.3) and occupancy graph mapping (section 4.8).

2.1.1.1. Occupancy Grids

Occupancy grid maps date back to the year 1989, when they were introduced by Elfes [Elfes 89]. Back then, the need for robust mapping and navigation capabilities of mobile robots was already obvious. The occupancy grid framework introduced by Elfes divides the state space into cells that store a probability of being occupied or free. This probability is estimated using Bayesian estimation and probabilistic models of the sensors.

A formal definition of occupancy grid mapping that fits the notations used in this thesis well is given by [Thrun 05, (9.1)] as estimating the map m using all measurements z and all robot positions x up to time t:

$$p(m \mid z_{1:t}, x_{1:t}) \tag{2.1}$$

In the following, the measurements z are regarded in a global coordinate system, i.e. they implicitly contain the robot pose x. This allows the problem to be stated as estimating the posterior probability over maps m using only the measurements z up to the time t:

$$p(m \mid z_{1:t}) \tag{2.2}$$

The key idea of occupancy grid mapping is to decompose the problem of estimating the *whole* map m into estimating single cells $m_i \in m$ by assuming conditional independence of cells [Burgard 08, 36.2.1]. Similar as in [Thrun 05, (9.4)], the posterior probability over the whole maps can be written as the product of posteriors of the single cells:

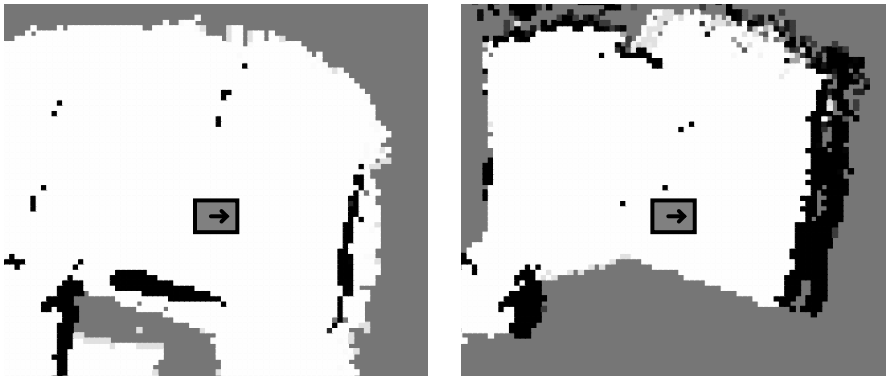
$$p(m \mid z_{1:t}) = \prod_i p(m_i \mid z_{1:t}) \tag{2.3}$$

For a single cell, the value $p(m_i)$ gives the probability of the cell being occupied, thus the occupancy grid mapping is a probabilistic algorithm and the resulting map is

[1] Graphs in the mathematical sense, i.e. structures consisting of vertices and edges.

not a deterministic representation of the world, but encodes the uncertainty about the current state. The core of the algorithm is a Bayesian filter with a (marginalized inverse) sensor model that describes the confidence of the sensors used for mapping. Together, both are used to update the posterior of cells $p(m_i \mid z_{1:t})$ if new sensor measurements are made. Bayesian filters as well as sensor models are further discussed in chapter 4, which covers probabilistic state estimation.

An example of two local occupancy grid maps which have been generated using the laser rangefinder and the RGB-D camera of the mobile demonstrator robot Artos (as described in details in section A.2) is given in figure 2.1.



(a) Created using the laser rangefinder **(b)** Created using the RGB-D camera

Figure 2.1.: Example of an occupancy grid map. White indicates certainly free, black indicates certainly occupied space. Gray is "unknown" (i.e. cells with a probability of 0.5). Data acquired using the mobile robot Artos (see section A.2) in the premises of the Robotics Research Lab (see section A.3).

The concept of occupancy grids becomes relevant later for visualization purposes and also in section 4.8, where the concept of occupancy *graph* mapping is introduced. This section also contains details about the update of the probabilities of occupancy grid maps. For this reason, the update process is not described in detail here.

2.1.1.2. Topological Maps

In contrast to the previously introduced occupancy grid maps, where single cells geometrically correspond to areas in the environment, topological maps focus less on this exact geometrical structure. Instead, they rather describe the *interconnection* of different entities [Burgard 08, 36.2.3]. Those entities could be any "significant places", e.g. landmarks or intersections [Thrun 05, 8.2.2]. For this reason, graphs are often a suitable way to represent topological maps. Consider an undirected graph $G = (V, E)$ with a set of vertices V and a set of edges E. If the vertices represent Cartesian positions in the environment (e.g. two-dimensional position vectors $(x, y)^T$), then the

edge between two vertices represents a connection on which mobile robots (or other entities) can traverse the environment. The exact geometric shape of that connection in reality is not important, it is abstracted by the edge. It could (in the simple case) be a straight connection, but also a curve connecting the two positions of the vertices. An example graph can be seen in figure 2.2. In this example, the hallway is exemplarily represented by a rectangular vertex, although in reality, it might probably have an elongated shape, connecting the three rooms and the exit.

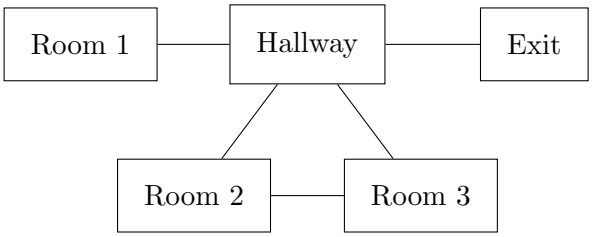


Figure 2.2.: Example of a topological map, represented through a graph.

Topological maps, in the form of graphs, are highly important in the subsequent chapters, where they are not only used to model the environment of the robot, but also to model the behavior of people in the environment, see section 4.3.

2.1.2. Localization

Mobile robot localization describes the process of determining the pose (the Cartesian position and the rotations with respect to a fixed reference coordinate system) of a robot. The knowledge of this pose is not always necessary in robotics. Some applications may not require the *absolute* pose of the robot, but may also perform adequate with less information. If for example a robot needs to traverse a narrow corridor, it may be sufficient to localize *relatively* to the corridor, to be able to not crash into the walls, but the absolute position along that corridor might not be interesting.

Still, robust *absolute* localization is needed for many applications, including the previously discussed process of mapping and the applications which are presented later throughout this thesis. Especially in global path planning using a known map of the environment, the current pose of the robot must be known, as it usually represents the start pose of the path planning task. Later, in the central chapters of this thesis, the use of sensor data, that is *not* originating from sensors that are mounted on the robot, is motivated and realized. In such a case, where sensor data from different positions needs to be fused, it is very important to really *know* the positions of all involved sensors as well as possible. Additionally, section 5.4 sketches a method to coarsely localize a mobile robot using radio signals from sensors distributed in the environment.

A mobile robot typically relies on one or more of its own sensors to localize itself in the environment. There are approaches that rely on additional technology in the environment to enable the localization process, but there are also procedures that do not require any special instrumentation of the environment.

2.1.2.1. Global Navigation Satellite and other active Systems

Into the first category fall methods like *Global Navigation Satellite Systems (GNSS)*, e.g. the very popular GPS system (initiated by the United States of America), GLONASS (the Russian equivalent to GPS) or the European Galileo system. These satellite systems usually only work properly outdoors and can only provide rough localization results (if possible at all) in indoor environments. However, there exist commercially available indoor localization systems that are to some degree similar to the GNSS systems. One example is the localization system by the company Ubisense, which uses ultra-wideband (UWB) radio signals to localize objects in buildings [Cadman 03]. The Ubisense system has e.g. been installed in the SmartFactory^{KL} [Meixner 10] where also the author of this thesis has conducted experiments.

In recent years, the iBeacon technology developed by Apple Inc. has received attention as an indoor localization system. It makes use of short-range Bluetooth low energy (BLE) beacons distributed in the environment that broadcast unique identifiers [Martin 14]. All such systems rely on additional infrastructure that has to be installed in the environment, in order to localize robots (or in general, other objects). In section 5.4, radio signals from wireless sensor nodes are used to provide coarse global localization. This is also an example of an *active* localization system.

The second family of localization methods does function without additional infrastructure. These localization methods can often also perform very well, but by principle, they may fail for the so called *wake-up robot problem* or the *kidnapped robot problem*[2], as they may not provide *global localization*. Global localization, according to [Fox 99a], refers to the process of localization without any initial information about the position.

2.1.2.2. Dead Reckoning

Probably the most simple method for localization is the process of *dead reckoning* or *wheel odometry*, as it is also often called. Dead reckoning calculates poses iteratively in small steps. As input, it uses small changes in the pose and sums them up. Of course, this process also accumulates errors and causes a characteristic drift between actual and estimated pose over time, making it unsuitable for long-term localization.

The input values are often directly coming from the ticks of the encoders mounted on a mobile robot's drive wheels, hence the name *wheel odometry*. Using the kinematic

[2] See [Fox 99a, p. 392]. For the *wake-up robot problem*, the robot must localize without any prior knowledge of the current position. The *kidnapped robot problem* is a special case of the first, where robot "knows" that it has been transferred to a different location.

model and parameters of the robot, the encoder ticks can be converted to linear as well as angular velocities of the kinematic center, which are then integrated by the dead-reckoning algorithm. It should be noted that the process of dead reckoning is not limited to using wheel encoders as inputs. An *inertial measurement unit (IMU)* can also be used to provide inputs for the dead-reckoning localization, and so can many other sensor systems that measure pose *changes* over time (see e.g. [Dudek 08, 20.1] for such applications). As already mentioned, dead-reckoning accumulates errors over time (or over the distance), so that the error grows large very quickly, if not being corrected regularly. The application of pure dead-reckoning is thus quite limited. Additionally, it cannot provide global localization, as there is no way to recognize absolute poses by the dead-reckoning process. This usually needs to be done manually when the robot is powered up. Please see figure 2.3 for a visualization of errors associated with wheel odometry. In the depicted sequence, a mobile robot was driving from a start point (in the western part of the figure, where blue and red meet) through the environment, back to the start point and once more through the environment. The blue arrows represent the pose estimations from a Monte Carlo localization (see below for more information). The red arrows are the calculated poses from the wheel odometry. It can be clearly seen that these poses do not represent reality, as they lead through walls and other obstacles in the environment.

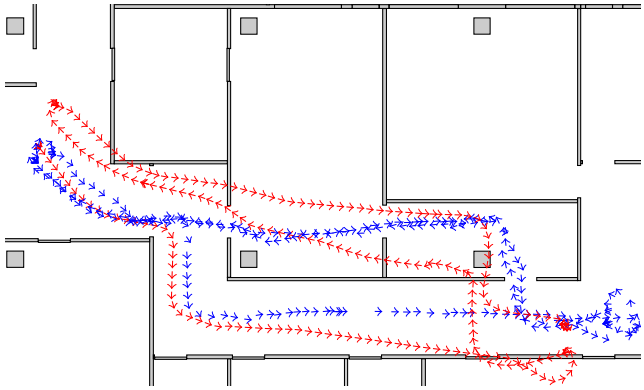


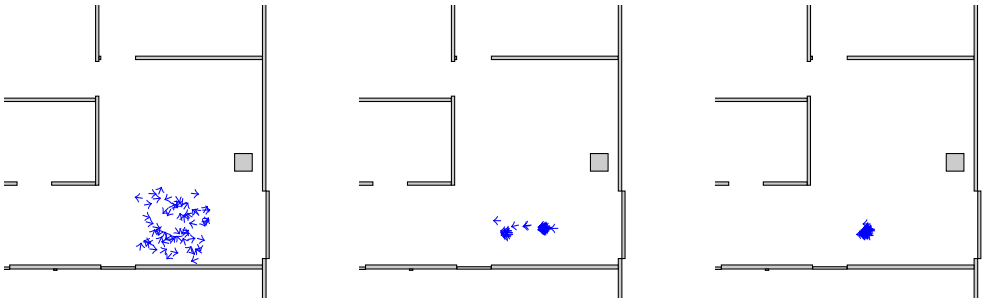
Figure 2.3.: Comparison of localization by wheel odometry (red arrows) and localization using Monte Carlo Localization (blue arrows). Data acquired using the mobile robot Artos (see section A.2) in the premises of the Robotics Research Lab (see section A.3).

2.1.2.3. Monte Carlo Localization

Monte Carlo localization (also called particle filter localization or Markov localization [Fox 99a], [Thrun 05, 7.2]) uses probabilistic algorithms to localize a robot using sensor data and a known map of the environment. Roughly speaking, distance measurements of the robot's sensors are matched with poses in the known map. This requires sensor

models (that describe the characteristics of the used sensors) and a probabilistic filter (often in form of a particle filter). As probabilistic filters play an important role in this thesis, they are discussed in detail in chapter 4. If the robot moves (as determined by dead reckoning), this information is usually also fed into the localization algorithm to improve the process or to localize in case there are no new sensor measurements (if e.g. the sensors are slow, compared to the movement of the robot). The estimation results are called the *belief*.

Monte Carlo localization can in principle solve the global localization problem by initializing the belief uniformly (over the whole map) at the beginning of the localization process. However, at the beginning there may be ambiguities. It usually takes some sensor measurements and robot movements till the belief converges to a single pose in the map. Usually, particles are used to represent the state (i.e. the pose) of the robot in the environment. In contrast to other representations of the belief, they have the advantage of not restricting the shape of the belief in any way. They can especially represent multimodal distributions with several candidate poses. See figure 2.4 for an example sequence of the Monte Carlo localization using particles. At the beginning, the belief is initialized uniformly in an area of about $2 \times 2\text{m}^2$ with different orientations. As the algorithm runs and sensor measurements are incorporated, the belief quickly converges to the real pose of the robot with some residual uncertainty.



(a) $t = 0\text{s}$ **(b)** $t = 0.31\text{s}$ **(c)** $t = 1.29\text{s}$

Figure 2.4.: Convergence of Monte Carlo localization after uniform initialization of particles in a square area. Data acquired using the mobile robot Artos (see section A.2) in the premises of the Robotics Research Lab (see section A.3). For the sake of visualization, only a fraction of the total number of particles is shown.

Regarding the wake-up or kidnapped robot problem, the localization process can easily be assisted and sped up if there are methods available to get a coarse initial localization. In this case, the coarse and rough estimate can be used to initialize the particles mainly in that region, so that the Monte Carlo localization can converge more quickly. Methods providing such coarse location estimates could be the (very weak indoor) signals of a GNSS or other methods based on wireless signals that e.g. use WiFi access points or a wireless sensor network to narrow the possible locations

of the receiver down. This application is discussed later in section 5.4, in the chapter about safe and cost efficient navigation.

2.1.3. Path Planning

Mobile robot path planning describes the problem of finding a sequence of robot configurations (poses) so that the robot is able to move from a start to a goal position. Usually, this path should be optimal with respect to some metrics (often it should be the shortest or fastest possible path). Additionally the paths should be collision-free, i.e. the robot should not hit any obstacles while driving. The topic is resumed later in section 5.2 and section 5.3 where the estimated state of the environment is used to improve the path planning of mobile robots.

Path planning can be divided into the tasks of *global path planning* and *local path planning*. In the literature, there are numerous definitions of the two sub-problems available, making the distinction between local and global not always easy. A very early definition was given by Crowley in the year 1985 [Crowley 85]. He defines global path planning as something that "requires a prelearned model of the domain which may be a somewhat simplified description of the real world" [Crowley 85, p. 31]. This prelearned model could e.g. be a map that represents the environment. Local planning (or local navigation or local obstacle avoidance, as he calls it) "carries out the steps in the global plan" [Crowley 85, p. 31] and "requires a model that reflects the state of the environment". This means that the local planner relies on the actual sensor data the robot is perceiving, while the global planner does not need them in his definition. He acknowledges that the process of planning local paths is needed to avoid obstacles (that are not known in the global map and are thus unexpected).

Another definition of local and global path planning can be found in the work by Singh et al. [Singh 00]. In their work, they also decompose the navigation problem into global and local planning stages. The purpose of their local planner "is to react to sensory data as quickly as possible avoiding hazards of various kinds" [Singh 00]. Their global planner is "more deliberative [...] operating at a coarser resolution".

Quinlan and Kathib [Quinlan 93] also emphasize that global path planning (or in their terminology just *path planning*) uses models of the world to plan a path to the goal. They say that moving along that path and handling small disturbances or unexpected obstacles is the task of the control system, which they do not explicitly call a *local planner*, but which is consistent with the above definitions of local planners.

Behnke also gives a good and concise definition in his work about multiresolution path planning [Behnke 04]. Local methods "do not attempt to solve the problem in its full generality, but use only the information available at the moving robot to determine the next motion command" [Behnke 04, p. 332]. He then continues to state that "global methods assume complete knowledge about the world".

In this thesis, the notions of global and local navigation are defined and used as follows: Global navigation uses known information such as a map or a graph to find a

path from a start position (which is usually equal to the currently localized pose of the mobile robot) to a goal position somewhere in the environment. Local navigation aims to follow the path given by the global navigator as close as possible. If it is not possible to follow that path exactly, either due to obstacles that were not known in the global representation of the world or due to kinematic constraints of the robot, the local navigator may deviate from the global path. The local navigator, as defined here, does thus also handle the problem of obstacle avoidance.

2.2. Aware or Smart Environments, Ambient Intelligence

Aware or smart environments are ordinary environments with added technology that has the primary goal of assisting the user. Often, this technology is not obviously visible to the people in the environment, it is thus *ambient* technology.

It is important to note that the term "smart" (at the time of writing this thesis) is used rather inflationary. In earlier research, it often refers to interaction technologies. Harper, for example, notes that what makes a house smart "is the interactive technologies that it contains" [Harper 03, p. 2]. In contrast, nowadays, it is often used to indicate that something is networked and possibly connected to the Internet, as stated by [Prassler 08, 54.3].

This definition matches many of today's commercially available products – e.g. one can buy *smart* phones, *smart* TVs, *smart* watches, *smart* scales, *smart* coffee machines and many more *smart* devices or appliances of everyday life. Common to most of these devices is not their built-in "intelligence", but rather the fact that these devices are connected to the Internet and that they can either access remote information, their functionality can be accessed from remote places or that they can publish information to remote places ("the cloud").

The notion of "smart" degrades more and more to a marketing term which continuously loses the meaning it had in earlier literature. This is the main reason why the author of this thesis has decided to use the term "smart" only in small doses and to use more appropriate descriptions instead. For example, the attribute "aware" for an environment emphasizes the fact that the environment is able to perceive what is happening in it through sensors, but does not require the house to be connected to the Internet.

There are many examples for aware environments in different domains. The most famous ones are probably those of the *smart homes*, i.e. environments where people live. Examples include the *Aware Home* [Kidd 99], the *Gator Tech Smart House* [Helal 09] or the *Robotic Room* at the University of Tokyo [Sato 96]. Not only the environments in which people live, but also where they work have been researched, in the form of *smart offices* [Gal 00]. Another example of aware indoor environments are so called *smart factories*. One example is the SmartFactory^{KL} [Lucke 08], [Zuehlke 08], [Meixner 10] at the DFKI Kaiserslautern.

So far, only indoor environments have been discussed, but do aware environments really have to be limited to being indoors? Obviously not, as research work regarding pedestrian-aware street lighting [Müllner 11] and the monitoring of traffic conditions [Mohan 08] show. While the following chapters of this thesis focus on indoor environments, the concepts that are suggested are not limited to only these indoor environments, but could be applied to the other domains as well.

There are numerous services that aware environments can offer to their inhabitants. Many of them are located in the field of *ambient assisted living* (AAL). The concept of ambient assisted living aims to help elderly people living in their homes through ambient technology [Wichert 12]. Examples include the detection of emergency situations and the automatic request for assistance in these situations. Other applications of aware environments are e.g. the increase of the comfort for the inhabitants and energy saving. An aware home could e.g. automatically switch on lights and heating, only in rooms where people are present (or will be present soon).

Most of these services require knowledge about the location of the inhabitants. Also, the work in this thesis is built upon this information. For this reason, the following sub-sections mainly focus on the detection and localization of human beings by aware environments. As the interconnection of devices raises security and privacy questions, these are discussed afterwards.

2.2.1. Detection and Localization of Human Beings

The detection of people and the identification of their exact location, as well as the tracking of their movements are fundamental parts of aware environment research (and are also quite important for the further chapters of this thesis). In this section, a brief overview about different sensors and methods that are used to detect people is given.

Most of the early systems for localization of people were *active* systems, that needed the subjects to be localized to wear some kind of device that was part of the localization process. These devices are sometimes called (active) *badges*. However, it has soon been noticed that active systems do not work if the people in the environment accidentally or deliberately "forget" to wear the badges. Thus, the concept of *device-free* or *passive* localization has emerged [Youssef 07]. This section first handles active, afterwards passive localization systems.

One of the earliest (if not *the* earliest) research projects about localization of people in smart environments is the Active Badge system developed between the late 1980s and the early 1990s at the Olivetti Research Laboratory (ORL) in Cambrige, England [Want 92]. The authors envisioned a system that could help other people knowing the location of their colleagues. They name hospitals, where, in case of medical emergencies, it is important to localize certain people quickly and present an example scenario where a receptionist can more easily locate people with the help of the Active Badge system.

As the name already hints, the system is *active*, which in this case means that people need to actively wear badges that also actively transmit infrared signals to stationary receivers installed in the environment. Authorized users can then view the current locations of all people through a central service.

It is very interesting to note that privacy concerns were quite big, back in 1992, when the original work was published. The authors report that initially, there have been many doubts and privacy concerns by the staff of the lab, in which the system was to be introduced. However, after some time, the benefits of the system surpassed the concerns and the system was accepted [Want 92, p. 99]. Nevertheless, they later discuss in detail that it is of course possible to use the technology not only for good, but also for malicious purposes and demand protection of the individuals' rights, should it be necessary [Want 92, p. 100].

The Cricket localization system [Priyantha 00] is an active system in the way that it requires the users to carry a Cricket device. The devices themselves (*listeners*) however, are *passive*, because they do not emit any signals. Localization relies on a combination of radio frequency and ultrasonic signals which are emitted by fixed *beacons* installed in the environment. Due to this combination, the system is immune to indoor radio phenomenons such as multipath propagation that might impair the localization process.

In contrast to e.g. the Active Badge system, the Cricket system can preserve the privacy of its users, because the *listeners* carried by the users are passive and the localization happens locally. Although the system is designed to be used with a central location discovery service, the decision to advertise an estimated position is made at the receiver and can thus be prevented. The authors emphasize that the goal of their system is "location-support", rather than "location-tracking" [Priyantha 00, p. 32].

Moving on to completely *passive* localization systems that do not require the people to wear any devices or tags, there exist a couple of interesting approaches. Besides coining the term *device-free localization*, Youssef et al. from the University of Maryland present a system that uses changes in received signal strength of WiFi devices to localize people [Youssef 07]. The authors envision applications such as intrusion detection and tracking in homes and offices, or asset protection in general, even at a larger scale by e.g. protecting borders or railroad tracks and therefore enhancing traditional security systems [Youssef 07, p. 222]. In contrast to similar systems, such as WiTrack [Adib 13] and WiSee [Pu 13], which both need special radio hardware, the system does not rely on special hardware, but works with standard 802.11 WiFi hardware, such as access points and wireless clients [Youssef 07, p. 224], thus reducing cost and installation effort.

In their work, Youssef et al. describe three main aspects – the detection of events, their tracking and the identification of the entities (i.e. people). Out of these three, the first two are tackled in the paper. According to their results [Youssef 07, pp. 225-227], the detection of events can be very reliable (depending on how the parameters used in the algorithms are chosen) and tracking of people using a previously learned radio

map results in decent position estimations, as far as the experimental results can be interpreted. The identification of entities using signal strength measurements however, is not dealt with in that work.

2.2.2. Security and Privacy

If technology is networked (especially when connected to the Internet), security of the systems becomes a very important topic. Measures must be taken so that the system cannot be abused by adversaries. This is even more crucial, if *wireless* technology is employed, which is very often the case with smart or aware environments' technology.

Additionally to this issue of *security*, also the *privacy* of people with respect to smart technology is important. Whenever personal data of people is collected and processed, care should be taken to protect that data from unauthorized access, or – even better – to avoid the collection of such data by design from the start.

In his article "Security Issues in Ubiquitous Computing" [Stajano 10], Stajano boils the issues with smart environments related to security down to an essence. According to him, just as with information systems in general, the threats to ambient systems affect three major aspects – confidentiality, integrity and availability.

Data acquired in smart environments should be *confidential*, as it may contain information that is private to the inhabitants of the environment, e.g. location data. Regarding *integrity*, the question to ask is: What happens if the data sent in a smart environment becomes corrupted, either accidentally or voluntarily by an adversary? The more people are relying on information systems, the more important the *availability* of the systems becomes. What does it mean for a smart environment to become unavailable?

Most of these threats are relevant to the work that is presented in the rest of this thesis. In the later chapters, the topic is resumed several times. In chapter 6, especially with regard to the proposed overall system.

3. Robots in Aware Environments

As the important fundamentals of mobile robotics and aware environments have been described in the previous chapter, this chapter now goes one step step further and introduces some works that are highly related to the topic of this thesis, as they handle both robotics and aware environments and in particular mobile robots *in* aware environments. There exist several completed research projects that fit into this chapter. These are handled first. However, there are also some smaller works of relevance that are mentioned after the larger research projects.

At the end of this chapter, based on the findings in the literature, the concept for safety, cost-efficiency and performance of mobile robots in aware environments, that has already been sketched in the introduction, is explained in detail.

3.1. Related Research Projects

In this section, related research projects are introduced. It starts with the *Physically Embedded Intelligent Systems* project, which, to the author's knowledge, is one of the earliest and one of the most influential related projects. It then continues with the *Robotic Ubiquitous Cognitive Network* project, the *ROBOT-ERA* project and the *Ubiquitous Networking Robotics in Urban Settings*.

3.1.1. Physically Embedded Intelligent Systems

The idea of having ecologies of *physically embedded intelligent systems* (PEIS[1]) originates from a collaboration between the AASS Mobile Robotics Laboratory at the University of Örebro in Sweden and the Electronic and Telecommunication Research Institute, South Korea. The first publication regarding PEIS dates back to 2005, where the idea of combining autonomous robotics with ambient intelligence is introduced [Saffiotti 05]. The research on PEIS is settled at the intersection of the fields of robotics, artificial intelligence and ubiquitous computing [Saffiotti 05, 2].

The devices that qualify as PEIS are diverse. The authors name robots (such as autonomous vacuum cleaners), sensors in the environment to localize objects and also "non-intelligent" things such as parcels which have been equipped with electronics [Saffiotti 05, 3]. In a later work by [Saffiotti 08, VI], more devices are named, e.g. a

[1] Pronounced /peɪs/, as explained by [Saffiotti 06].

table with an RFID reader and remote-controlled lamp. Peis could be anything "as simple as a toaster and as complex as a humanoid robot" [Saffiotti 05, 3].

Peis are no isolated devices, but they form an ecology. While a single Peis is defined as a "set of inter-connected software components residing in one physical entity" [Saffiotti 05, 3], a Peis-ecology is a "collection of inter-connected Peis, all embedded in the same physical environment" [Saffiotti 05, 3]. In a Peis-ecology, Peis can make use of each other's functionalities and the power of the ecology emerges from the cooperation and the interaction of single Peis.

The authors explicitly state that instead of focusing on a complex, isolated human-like robot to assist people, the robot should merge into the environment, by using the (smart) environment's perception as well as actuation capabilities [Saffiotti 05, 1], similar as the idea of the "disappearing computer" (as e.g. described in the preface of [Streitz 07]). The authors name an example in which a mobile vacuum cleaning robot is augmented with global localization through cameras distributed in the environment [Saffiotti 05, 3]. In a different example, two robots work together on a task of pushing a box through a door [Lundh 06]. In this example, the second robot provides relative localization for the first robot, as its local sensors may be obstructed by the box.

This cooperation of single Peis requires communication, which is achieved through a middleware [Saffiotti 05, 1], [Broxvall 06b, III.]. This middleware is called the Peis-kernel [Saffiotti 05, 4.2] which can be run as a run-time library on Linux systems and establishes peer-to-peer communication through TCP/IP. As the middleware must be suitable for robots and also small embedded devices, there is also a version of the Peis-kernel for TinyOS, to be run on small devices with limited processing power and memory [Bordignon 07]. Instead of TCP/IP, this implementation uses IEEE 802.15.4 communication with peer-to-peer networking primitives of TinyOS [Bordignon 07, IV. B.]. There exists also an alternative implementation (besides the previously mentioned reference implementation) of the middleware written in Java [Seo 06].

The task of the middleware is not only to establish communication, but also to enable self-configuration [Saffiotti 06, 4.3] and dynamic reconfiguration of a Peis-ecology [Saffiotti 06, 4.4]. The exchange of knowledge between Peis is realized using tuples in a distributed tuple space [Saffiotti 05, 4.2]. Each tuple contains the identifiers of the Peis itself as well as the one of the local component inside the Peis, and a number of key-value-pairs to store actual data. The keys are strings, except for the TinyOS implementation of the Peis-kernel, where they are integers [Bordignon 07, IV. B.]. The knowledge about the semantics of the tuples is stored in the Peis-ontology, which is itself held as tuples in the tuple space.

Besides the reference architecture and the already mentioned examples for Peis, the researchers around Peis have built the Peis-home, a sensorized environment to conduct experiments with Peis [Saffiotti 05, 4.1]. The concept of Peis-ecologies has been evaluated in several experiments. The first publication, [Saffiotti 05], describes an example with a total of four Peis: Two robots, a localization system using cameras

in the environment and a visualization/debug monitor. In this example, one robot delegates a task to the other robot and the localization system is used (whenever possible) to localize the robots.

The application presented by [Broxvall 06a] aims to recognize odors using a Peis-ecology. The ecology consists of several simple gas sensors, located at tactical positions (e.g. in the fridge) in the environment and a robot with a more complex gas sensor. If one of the simple sensors raises an alarm (for example a milk going bad in the fridge), then the robot can be sent to that position to investigate the situation further using its sophisticated odor sensor.

The Peis-ecology by [Bordignon 07, V.] was created in the context of the TinyOS implementation of the Peis-kernel. It consists of a mobile robot running the full Peis-kernel and three small devices that run the tiny Peis-kernel. The experiment mainly shows the successful interaction with, and the seamless integration of embedded devices into the Peis-ecology.

The final demonstration experiment of the Peis project is described by [Saffiotti 08, VI.]. In that experiment, run in the abovementioned Peis-home, one robot, a tracking system, a table with RFID reader, a lamp and the monitoring station (also bridge to the IEEE 802.15.4 network) are the relevant Peis. The example run described in that work is (with some abbreviations) the following: A user gives the instruction to bring him a book to his bed. The robot gets the task to fetch the book and first finds out the position of the book (on the table) using its RFID tag. The robot then switches the light on, fetches the book (while localizing using the external tracking system) and delivers it to the user.

By regarding them as ecologies, the Peis project has delivered some impressive results concerning robots in aware environments. The most important result of the project is probably the Peis-kernel, which provides the *middleware* for the interaction of different Peis. In the presented experimental results, it has shown to be very *versatile*. It is no surprise, that this kernel is also used in the other two major projects that are described in a moment, the RUBICON and the ROBOT-ERA projects. In contrast to this thesis however, there was no clear focus on properties to be optimized by embedding robots into Peis ecologies. In this thesis, these properties shall be clearly defined (as safety, performance and cost-efficiency), as developed at the end of this chapter.

3.1.2. The RUBICON Project – Robotic UBIquitous COgnitive Network

The RUBICON project can be considered a follow-up of the research on Peis. One of the leading people in Peis, Alessandro Saffiotti, was also involved in the Rubicon project. Also, many of the goals are similar to those of the Peis ecologies. The RUBICON architecture makes use of the Peis middleware as a communication layer [Amato 12a, 2].

One of the main goals and an improvement over the "classical" robot ecologies, like in the Peis-project, is to have components that are self-learning to reduce the amount

of pre-programming prior to deployment, to reduce costs and to make installation more simple [Dragone 11, p. 42]. Further goals are improved quality of service in Ambient Assisted Living scenarios and adaptation to dynamic changes in resources (i.e. robots, sensor/actuator nodes and appliances) ultimately leading to a large scale adoption of these technologies [Dragone 11] and a reduction of costs and maintenance effort [Amato 15, 1]. The authors believe that their "self-sustaining learning solutions [...] [will yield] cheaper, adaptive and efficient coordination of robotic ecologies" [Amato 12a]. The project aims to make use of "methods from cognitive robotics, agent control systems, wireless sensor networks and machine learning" [Amato 12b].

The authors of [Amato 12a] state that having multiple robots in a smart environment reduces complexity and increases individual values due to new services that could not be performed before [Amato 12a, 1]. RUBICON ecologies make explicitly use of wireless sensor network technology combined with robotics to reduce the complexity of either of the devices and to enhance their value [Bacciu 12, 1]. Also, matching the already-stated goals of RUBICON, the "classic" wireless sensor networks should be enhanced by learning-capabilities [Bacciu 12, 2] to further enhance them.

The architecture of RUBICON consists of several layers [Amato 15, 3]. There exists a control layer, a learning layer and a cognitive layer. All of them are connected through the communication layer. Details can be found in [Amato 15, 3].

The application of mobile robot navigation is discussed by [Bacciu 14a]. The work shows an approach to come up with better plans for mobile robot navigation by making the planner context-aware. A mobile robot which uses mainly its RGB-D sensor for localization is used in a demo environment where the sensor is either impaired by a mirror in the room (depending on the chosen path) or by the fact that the user switches off the RGB-D sensor for privacy reasons. In both these cases, the authors show improvements in the quality of the navigation when employing the knowledge that has been learned from previous runs.

A framework based on OSGi, that helps to specify and program robot ecologies is presented by [Dragone 13]. In an example presented in that work, the authors show how, after a sensor failure of one robot, the localization of this robot is achieved using the help of a second robot. Localization (mainly of people) in the ecologies is handled e.g. by [Barsocchi 11], where received signal strength measurements between fixed and mobile wireless sensor nodes together with context information are used to localize mobile nodes.

The work by [Bacciu 11] describes a system that is able to predict the movements of people within an environment outfitted with a wireless sensor network. In their experiments, four mobile sensor nodes (anchor nodes) have been placed at fixed positions in the environment and a fifth sensor node was being carried by the person. Methods based on the reservoir computing paradigm have then been used to predict the movement of that person using a sequence of received signal strength measurements between the mobile node and the anchor nodes. The work has been extended in a later publication by [Bacciu 14b].

Final testing during RUBICON was done in a smart-home apartment (called the HomeLab) of 45m^2, providing living-room, kitchen, bedroom, bathroom and a corridor [Amato 15, 4.1]. It was outfitted with several types of sensors (motion, pressure mats, microphones and others), actuators (blinds, lights, door locks, appliances), a mobile robot and wireless sensor nodes [Amato 15, 4.1]. In the experiment, first, training runs have been recorded with more than 50 sensors as inputs [Amato 15, 4.2]. The ground truth (the activity of the experimenters, e.g. sleeping, eating, etc.) was manually annotated. To test the learning layer, the system has been trained to classify different activities based on different sensor streams [Amato 15, 4.3]. The activity recognition was then tested on a test set. The cognitive layer was tested in another experiment [Amato 15, 4.4] where it was trained that the television set should be turned on whenever the user eats and to clean up the floor using a robotic vacuum cleaner when the user has finished eating and starts washing the dishes.

The RUBICON project focused on self-learning technologies to enhance robotic ecologies. In contrast to the Peis project, it also explicitly *had* the goal to reduce costs, but only those caused by the initial idea *itself*, the combination of mobile robotics with ambient intelligence and the associated efforts in deployment and maintenance. Also, some rudiments of increasing performance through adaptation of the systems are visible. Still, the focus of the RUBICON project lies on technologies for self-learning and not on increasing performance or safety, unlike the goals of this thesis.

3.1.3. The ROBOT-ERA Project

The name ROBOT-ERA is short for the title "Implementation and integration of advanced Robotic systems and intelligent Environments in real scenarios for the ageing population". Participants of the project included, among others, also the University of Örebro (which was also affiliated with the Peis ecologies and the RUBICON project).

The main goal of the project was to "enhance the quality of life and independence of elderly people through robotic assistance" [Broz 12, Abstract]. The authors also explicitly name the integration of robotic services with "intelligent environments". The ideas of the project are motivated by the projection of an increase in the amount of people aged 65 and above in Europe [Cavallo 12]. It aims to help this age group in the future using robotic systems and ambient intelligence. Additionally, the appearance and acceptability of robotic systems by elderly users was a goal to be studied [Cavallo 12].

Services to be offered by the ROBOT-ERA project to the inhabitants are "shopping delivery, reminding, communication, laundry, food delivery, object transportation and manipulation, garbage collection, surveillance, indoor escort [and] outdoor walking support" [Bevilacqua 15, 1.1]. In [Rocco 14], the authors claim to be the first to have created an autonomous multi-robot system in a smart environment servicing elder people [Rocco 14, p. 10]. As sensors in the smart environment, they have used wireless sensor nodes equipped with passive infrared, pressure, humidity, light or temperature sensors [Rocco 14, 4.1].

One of the robots developed for the ROBOT-ERA project, the *domestic robot*, is described in detail by [Hendrich 14]. It is a platform with differential drive kinematics and a manipulation arm. It uses several sensor systems, among others an RGB-D sensor, laser rangefinders and a high-resolution camera [Hendrich 14, III.]. The software control system uses a combination of different frameworks and middlewares. Besides the Robot Operating System (ROS), it also employs the Peis middleware and a proprietary navigation stack [Hendrich 14, IV.]. Other robots that are part of the ROBOT-ERA project are (apart from the already mentioned *domestic robot*) the *condominium robot* and the *outdoor robot* [Bevilacqua 15, 1.1]. The condominium robot is able to navigate a house through an elevator, the outdoor robot can be used in urban environments and has a cargo container for delivering payloads.

[Sathyakeerthy 13] motivates the idea of not only having single robotic ecologies, like in the Peis-project or the RUBICON-project, but to take the idea one step further to have *multiple* robot ecologies that span whole neighborhoods and towns. According to the authors, this introduces new challenges such as visibility and aggregation of services beyond in these larger ecologies.

Experiments have been executed in the DomoCasa Living Lab of the Biorobotics Institute in Italy and in the Ängen Smart Home in Sweden [Rocco 14, 5]. In one, a person orders food and two robots (one indoor, one outdoor) synchronize to bring the food to the user [Rocco 14, 5.1]. The indoor robot is capable of using the elevator in the house to deliver the goods to the correct floor. The outdoor robot is able to fetch the prepared order from the store.

The ROBOT-ERA project indeed demonstrated the extensibility of the concepts of a single robotic ecology (as known from the Peis and the RUBICON project) to a large-scale ecology of ecologies, spanning more than a single home. However, it has lost some of its focus on the interaction of mobile robots and smart environments, compared to the previous two projects.

3.1.4. Ubiquitous Networking Robotics in Urban Settings

The objectives of the *Ubiquitous Networking Robotics in Urban Settings* (URUS) project are outlined by [Sanfeliu 06]. Its main goal was to design a network of *robots* (in the context of the project, this term explicitly includes not only "classic" robots, but also intelligent sensors and devices) that interact with humans in the environment. As intelligent sensors, the authors name video cameras and "acoustic sensors", intelligent devices could e.g. be mobile phones. As scientific challenges, this first publication lists some tasks that are related to the goals of this thesis. Among others, those are cooperative environment perception, cooperative mapping and cooperative task negotiation.

It is also interesting to note that the authors aim to *efficiently* achieve tasks "that in the other way can be very complex, time consuming or too *costly*" [Sanfeliu 06, II]. With the "other way", the authors mean the traditional way without cooperation

between networked robots. The term *network robots*, is defined by the authors as a "new concept that integrates robots, sensors, communications and mobile devices in a cooperative way" [Sanfeliu 06, II].

One work created in the scope of the URUS project describes an attempt to solve the global localization problem for mobile robots in (large) cooperative environments, such as the urban setting of the URUS project [Murtra 08]. Using a known map, sensor data from local sensors and also data from other robots as well as sensors in the environment, a mobile robot should be enabled to improve its global localization[2] strategy. In simulation experiments, the authors show that the strategy chosen for localization varies for traditional ("non cooperative") and cooperative environments. Unfortunately, they do not evaluate (e.g. in terms of the time needed to finally localize globally) the improvements due to the cooperative environment.

Another very interesting publication in the context of the URUS project is the one discussing legal issues, especially in the context of the privacy of inhabitants of the proposed environments [Sanfeliu 10]. The work raises the question how robots in public spaces should be legally regarded. If a robot in an (urban) network robot system captures the face of a person, this data is personal data of the depicted person, how should this be (legally) treated? Camera images are not the only personal data – voice recordings and other biometric data are further examples. Besides that, also location data is an issue, if the robots (or cooperative environments) are able to determine the position of people.

The URUS project is in some aspects highly related to this thesis. It explicitly names the properties *efficiency* and *cost* that should be improved through a network of cooperating robots and external sensors, compared to a scenario without cooperation. Both are two of the (three) key properties that shall also be optimized within this thesis. Additionally, the legal aspects discussed by [Sanfeliu 10] are interesting, because especially privacy in aware and smart environments is an important topic. As can be seen soon, maintaining the inhabitants' privacy using *sparse* sensor instrumentation of the aware environment is one of the goals of this thesis.

3.2. Other Related Work

There exists definitely more related work than just the projects that have been mentioned so far. A good overview is given by [Mastrogiovanni 10], published in the *Handbook of Ambient Intelligence and Smart Environments*. It summarizes the paradigm of *Ubiquitous Robotics*, which describes robots that are "able to interact with devices distributed throughout the environment and get across heterogeneous information by means of communication technologies" [Mastrogiovanni 10, I.]. The ubiquitous robots envisioned in the article have only limited perception and actuation capabilities and operate in "well-suited" environments.

[2] Global localization according to the definition in section 2.1.2.

Mastrogiovanni describes that this will make those systems *cheaper* and more *reliable*. To build more cost-efficient systems is also one of the goals of this thesis and more *reliability* could roughly be put on a level with *safer* systems, that are another goal of this thesis. However, the article does not explicitly mention it. The same applies for *performance*.

The concept of *Ubibots* originates from the Robot Intelligence Laboratory, KAIST, South Korea. It consists of three forms of robots: software robots (Sobots), embedded robots (Embots) and mobile robots (Mobots). All of them provide services to users in ubiquitous spaces [Kim 04]. The authors claim that their Ubibots constitute the third generation of robotics (with the first being industrial robots and the second being personal robots).

Software robots are, as the name suggest, purely software based entities and can thus move between different (hardware) systems. They are able to interact with users, e.g. in form of avatars. Embots are either residing in the environment itself or in Mobots. Their task in the system is to provide perception. For this reason, Embots are always equipped with sensing and networking capabilities. Finally, Mobots are mobile devices and able to provide various services to users through actuation capabilities. Further information about Ubibots, such as example Embots, Mobots and a Sobot are given by [Kim 07].

The *Artificial Ecosystem* originates from the Laboratorium Group, University of Genova, Italy [Mastrogiovanni 10, 2.4]. Robots in the Artificial Ecosystem work together with smart nodes distributed in the environment. These smart nodes have the tasks to assist robot localization, to detect emergency situations (such as a fire) and they can also have actuation capabilities by controlling automated doors or elevators. Autonomy in the Artificial Ecosystem can be seen as "a property distributed throughout the environment" [Mastrogiovanni 10, 2.4].

A definition of a *Symbiotic Robotic System* is given by [Coradeschi 06]. They define it as a system that "consists of a robot, a human, and a (smart) environment that cooperate symbiotically to perform a task". According to that definition, all of the previously introduced projects and also the work described in this thesis qualify as symbiotic robotic systems.

3.3. Concept for Safety, Cost-Efficiency and Performance

After having summarized the relevant related work regarding robots in aware environments, the concepts of this thesis are developed in detail in this section. Whenever applicable, the text refers back to the works of the state of the art and discusses its connections to the thesis' concepts.

As it has already been mentioned in the introduction, this thesis is built around the concept of *simultaneously* optimizing the three – usually opposing – design goals of *safety*, *cost-efficiency* and *performance* of mobile robots (see figure 1.1 on

page 1). The idea to reach this state is to shift the complexity away from local sensor systems of mobile robots into multiple sensors distributed in the environment. This is a similar idea as the one of the PEIS-project which aims to simplify autonomous robotics problems by "replacing complex on-board functionalities with simple off-board functionalities plus communication" [Saffiotti 06, 3], and the URUS project which also aims to increase efficiency and decrease cost by establishing cooperation between robots and the environment [Sanfeliu 06, II]. This shift, along with the design goal of maximizing all three parameters – safety, cost-efficiency and performance – forms the central thesis of this work, which shall be defined as follows:

> By consequently making use of *external* sensor data, it is possible to increase *cost-efficiency*, *safety* and *performance* of mobile robots.

This thesis aims to solve the above dilemma of having to choose between either performance or cost-efficiency (or the trilemma, if safety is also chosen to be subject to negotiation, which is usually not an option). The central thesis can be refined using some side theses. The first is about the origin of the external sensor data and it states:

> The *external* sensor data does not necessarily require the installation of *new* sensors in the environment. Instead, it shall be made use of *existing* infrastructure.

This is an important statement, as installing external devices into the environment will of course create additional costs and will thus initially *decrease* the intended cost-efficiency. The effort of installation and maintenance of robotic ecologies is also indirectly acknowledged by one of the publications of the RUBICON project [Amato 15]. If more than a single robot is operating in the environment, this installation effort may shrink, compared to the overall cost of the system, but nevertheless, this thesis states that *existing* infrastructure shall be used whenever possible. This existing infrastructure could be in the form of *smart* or *aware* technology that is able to provide information about people in the environment, e.g. motion sensors. It should be noted that none of the previously discussed related works does explicitly built upon this strategy. There are a couple of more side theses that are related to the type and the number of sensors in the environment as well as their properties:

> The sensor instrumentation of the environment may be *sparse*.

A *sparse* sensor instrumentation means that there may be only a few external sensors distributed in the environment. Still, in such a situation, the proposed system shall work. This assumption especially increases the difficulty of an exact multi-target tracking, as can be seen later in section 4.7. From this thesis, it is possible to go even further and require:

> The system shall be *fault-tolerant*, up to the *complete* failure of all external sensors.

This is a very important requirement. Unless very special hardware[3] is used, external sensors are prone to failure. Smart technology is often powered by batteries which may run low and will contribute to sensor failures. Wireless communication does also make things worse, as wireless communication can in principle easily be sabotaged, see section 2.2.2 on security in smart environments. The methods that are used to tackle the problem must thus anticipate failures and be able to cope with them. Probabilistic methods (see chapter 4) have shown to be a very adequate choice to handle these situations.

Even if the related works do not explicitly consider sparse sensor instrumentations, some of them discuss the fault-tolerance of the systems. The Peis-project mentions the need for dynamic reconfiguration: "A Peis-Ecology system should dynamically adjust its configuration to adapt to a changing environments and new situations." [Saffiotti 06, 4.4]. This feature should also introduce some fault-tolerance to Peis-ecologies, as a failing sensor (assuming the failure can be reliably detected) constitutes a changing environment. Additionally, in the scope of the RUBICON project, one goal is to "adapt to changes in the resources available, including [...] added/removed sensing/acting devices [...]" [Dragone 11, p. 43] to improve fault tolerance and reliability. Finally, there are two important statements related to the people living, working or generally acting in the environment.

> The people in the environment shall not be required to actively participate for the system to work.

This means that the people do not need to wear any *active* technology. Instead, only *passive* sensors will be installed that do not require the people to wear anything like e.g. active badges or tags. This is important for the safety of the system, as there must be no way for people to "sneak past" the sensors to avoid detection. If active participation is required, people could in such cases just (voluntarily or involuntarily) "forget" their badge or tag. This is again in contrast to other work, like e.g. the RUBICON-project where examples are given that localize people with a mobile wireless sensor node they *actively* have to wear [Barsocchi 11], [Bacciu 11, 2.2]. At the same time, the following is important:

> The system shall not invade the privacy of inhabitants directly or make use of any sensors that would violate the privacy of inhabitants.

This means that the sensors should be *anonymous*, i.e. the sensors should not be able to *identify* people. This is another aspect that makes some methods harder, especially the multi-target tracking which is introduced in the later chapters. Nevertheless, in the author's opinion, privacy is a very important good that is often neglected in current (at the time of writing) technology. While the Peis-project names privacy as one of the design goals [Saffiotti 06, 4.5], this aspect is unfortunately underrepresented in most of the other publications about Peis. Especially the camera-based localization

[3] For example safety-certified components from automation engineering, connected using real-time capable bus systems. Obviously, such components increase the overall system costs.

system for the robots, named and used in several Peis examples and experiments [Saffiotti 06, 2], [Broxvall 06a, IV. B], [Broxvall 06b, II. B], [Saffiotti 08, II. A] clearly raises privacy concerns. Fortunately, at least the URUS project takes privacy more seriously, as can be seen from the discussion of legal issues [Sanfeliu 10]. Nevertheless, even this project makes use of rich, identifying sensors such as cameras.

The use of external sensor data from aware technology presents challenges for the further processing of that data. Sensor failures are common and must be anticipated. The (possibly) sparse instrumentation of the environment leads to unobservable areas, yet their state needs to be estimated to some degree to be able to efficiently make use of the data. Chapter 4 establishes foundations to cope with these challenges by introducing *probabilistic* methods to estimate the environment's state using partial and error-prone sensor measurements.

In sections 4.1 and 4.2, state estimation in general and probabilistic state estimation based on recursive Bayesian estimation in particular are introduced. Section 4.3 describes the state space used for estimation within this thesis and section 4.4 describes the motion model which represents a probabilistic description of the behavior of the people in the environment.

Because of the requirements of not relying on active participation of people in the environment and to maintain their privacy, the main sensor type used in this thesis is a passive infrared (PIR) sensor. This sensor can be used to detect movement of people and delivers only sparse information (motion or no motion). It also cannot be used to identify people directly. Section 4.5 describes in detail how this type of sensor is modeled, so that it can be used within the probabilistic framework.

The real application of increasing *cost-efficiency*, *safety* as well as *performance* of mobile robots is contained in the chapters 5 and 6. The former contains several applications that use the external data for obstacle avoidance (section 5.1), path planning (section 5.2 and section 5.3) and to assist robot localization (section 5.4). It also presents an extension of the system to other types of sensors, in this case mounted on another robot in the same environment (section 5.5). The latter, chapter 6, regards all single aspects in one large application study experiment.

4. Probabilistic State Estimation

The estimation of the state of the environment using probabilistic methods and sensor information is an important ingredient of this thesis. For this reason, this chapter gives an overview about the topic, with special focus on methods for estimating the positions of people in the environment, the so called tracking.

The chapter is structured in the following way: First, probabilistic state estimation in general, especially using recursive Bayesian estimation is introduced and the use of probabilistic methods for this work is justified. Different possibilities to represent the state of the environment (using so called state spaces) are evaluated along with models of the employed sensors and models for the behavior of people. The rest of the chapter examines the application of probabilistic state estimation to the goals of this thesis. It discusses the tracking of people in aware environments (first in the single-target, then in the multi-target context) and also introduces another method for estimating the state of environment, called the occupancy graph mapping, which is based on the ideas of occupancy grid mapping.

4.1. State Estimation

State estimation describes the process of estimating the state of a system using only partial information about it. The state of the system is assumed to not be *directly* observable, but instead, it can be observed using partial and possibly erroneous observations. These observations are usually provided by sensors that observe the environment. In general, a system can also be influenced by issuing controller values, but this aspect is of less importance for the application of this thesis.

4.1.1. Motivation for Probabilistic Methods

The use of *probabilistic* methods for state estimation in this thesis is motivated by a few properties of aware technology and the people that inhabit the environments. First of all, aware technology is not primarily designed to be fail-safe or very robust, but usually low cost is of higher importance. This is in particular true for the wireless sensor nodes that have been used throughout this work (see section A.1), where single sensors may fail very easily[1]. For this reason, the sensors embedded into the environment are inherently *unreliable*. They only provide correct values with some

[1] Nodes actually *did* fail several times during experiments.

probability. Thus creating probabilistic models of the behaviors of sensors that can then be used in probabilistic filters is a very convenient way to anticipate and overcome the limitations of this technology.

Regarding the *tracking* of people, there is another reason to use probabilistic methods in favor of others. Namely, not only sensors can be easily modeled, but also the behavior of people. As people usually behave "randomly" to a certain extent (of course, there are also patterns in behavior that are less random) a probabilistic description of their behavior is appropriate.

Probabilistic state estimation can be roughly divided into three main aspects. The definition of the states that are to be estimated – the *state spaces*, the modeling of sensor and system behavior using sensor and motion *models* and *algorithms* that perform the actual estimation. These ingredients are discussed in the following subsections.

4.1.2. State Spaces

The state space of a system or an object describes the possible configurations that it can attain. Usually, only the aspects of a system that are of interest for a specific application are modeled in the state space. For a two-dimensional mobile robot localization system, this state space could e.g. consist of either only the robot position in Cartesian coordinates $(x, y)^T$ or it could be modeled as the whole robot pose – i.e. additionally including the rotation of the robot, resulting in: $(x, y, \gamma)^T$. This example illustrates that state spaces are usually multi-dimensional. While it is in principle possible to have scenarios in which the state can be encoded by a single scalar variable x, in general, a whole vector is used[2] to encode the state: $\boldsymbol{x} = (x_1, \ldots, x_n)^T$.

At this point it should also be noted that the variables representing a state may be either *continuous* or *discrete*. Examples for continuous variables include the abovementioned Cartesian coordinates while a discrete variable could e.g. be the state of a door which is either "open" or "closed". Different probabilistic algorithms might impose different restrictions on how the state space is composed. For example, the discrete Bayes filter (which is introduced shortly) relies on discrete state spaces encoded using discrete variables. Others, e.g. the popular Kalman filter, work with continuous variables.

4.1.3. Model Knowledge

Model knowledge is very important for state estimation, as the models provide the knowledge to estimate states using only partial and noisy observations (measurements). The faulty or noisy measurements are handled using so called *sensor models* (or also called *measurement models* [Arulampalam 02, I], *perceptual models* [Fox 03] or *observation models* [Ryu 07, II]) that – simple speaking – describe how likely an

[2]For the sake of readability, in the following text, the vector-notation of states is omitted.

observation is in the current situation. The partial view on the system state is handled using *motion models* that predict the evolution of the environment, even if no new measurements are available. The motion models are sometimes also called *actuation models* [Thrun 02, 2], *dynamics* [Fox 03] or *system models* [Arulampalam 02, I]. Later in this chapter, typical models are explained and the ones that are used in this work are described in detail. While some applications of Bayesian state estimation require both models, the occupancy graph mapping described later is an example of a state estimation that only utilizes the sensor model.

4.1.4. Algorithms for State Estimation

Finally, algorithms that perform state estimation on the state space using the model knowledge are needed. Recursive Bayesian filters based on Bayes' rule are very popular in different domains. There are many instances of such filters, e.g. the discrete Bayes filter, the Kalman filter or particle filters. Because of their importance, they are handled in more detail in the next section. However, as can be seen later, there are also others algorithms that are used to estimate states despite those filters. For example, the concept of *occupancy grid mapping* is very popular in robotics and is later used as the base for the concept of *occupancy graph mapping*.

4.2. Recursive Bayesian Estimation

Generally speaking, the purpose of Bayesian filters is to estimate the state of a system that is not directly observable. Bayesian filters achieve this using the two kinds of models that have just been introduced: a motion model, which probabilistically describes how the observed object "behaves" and a sensor model, which (also probabilistically) describes how reliable the sensors work.

The process of Bayesian filtering or Bayesian estimation is *recursive*, as the current estimation is calculated using the previous estimation and newly observed measurements[3]. This recursive processing, which is the core of every Bayesian filter, is based on Thomas Bayes' famous theorem, the *Bayes' rule* or the *Bayes' theorem*, see (4.1).

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)} \tag{4.1}$$

The possible applications of Bayesian filtering are manifold. Solely in the field of robotics, there is a multitude of challenges that are elegantly solved with the help of Bayesian filters. Mobile robot localization (as discussed in the fundamentals, see section 2.1.2) is often realized using particle filters (which are introduced soon)

[3]This depiction of the recursive estimation process is simplified, as it does not mention that (if used in the motion model) the current control values also influence the estimation. This aspect is included in the formal definition shortly.

that perform map matching using range data and noisy, error-prone odometry measurements [Fox 99b]. One way to solve the popular Simultaneous Localization and Mapping (SLAM) problem is using an extended Kalman filter [Thrun 08]. Besides robotics, inertial navigation and very importantly, target tracking does also make use of Bayesian filters (see e.g. [Bar-Shalom 78, Isard 98, Särkkä 07, Sidenbladh 03, Vo 03], to name just a few works that employ Bayesian filters). Target tracking is resumed in more detail in section 4.6 and section 4.7.

4.2.1. Bayesian Filters

More formally, Bayesian filters are used to estimate the hidden variables of a *Hidden Markov Model (HMM)*. In figure 4.1, the scenario is depicted in form of a *Dynamic Bayes Network (DBN)*. Variables x_i, which represent the real states, are the hidden variables which should be estimated by the filter. They are influenced by the (visible) control variables u_i and can be indirectly observed through the (visible) measurement variables z_i.

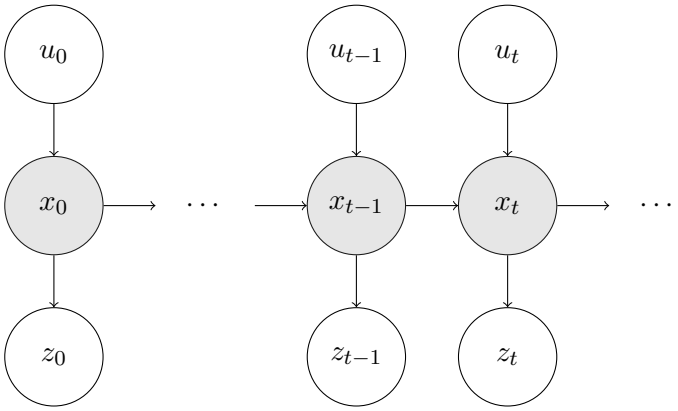


Figure 4.1.: Dynamic Bayes network to illustrate the process of recursive Bayesian filtering. Only the non-shaded variables of the network are directly observable (or controllable, respectively), the shaded ones are hidden. The arrows indicate conditional dependencies between variables. Figure originally created for [Arndt 11a], based on [Thrun 05, Fig. 2.2]

The recursive Bayesian estimation has the goal to estimates the state x_t of the environment at time t, given all the observations $z_{0\ldots t}$ and all the control values $u_{0\ldots t}$ from the beginning of the estimation process till the time t. The result of the estimation is usually called the *belief* over the state variable [Thrun 05, (2.33)] and is defined as given in (4.2):

$$\text{belief}(x_t) = p(x_t \mid z_{0\ldots t}, u_{0\ldots t}) \tag{4.2}$$

Using this, the recursive Bayesian update-rule can be written as in (4.3).

$$\text{belief}(x_t) = \eta\, p(z_t \,|\, x_t) \int p(x_t \,|\, u_t, x_{t-1})\, \text{belief}(x_{t-1})\, \mathrm{d}x_{t-1} \tag{4.3}$$

In some sources, e.g. [Thrun 05, Table 2.1], the process of the recursive Bayesian estimation is explicitly divided into the two steps of *prediction* (the inner part/the integral of (4.3)) and *correction* using new sensor data (the application of the sensor model $p(z_t \,|\, x_t)$). The value of η in the update-rule needs to be computed so that the resulting belief represents a probability density function, i.e. that the integral over the whole state space equals one.

The recursive Bayesian update rule given in (4.3) can already be considered a probabilistic filter. However, it can unfortunately be seldom used directly, because in general, there might exist no closed-form solutions of the rule itself or even of the models and the belief. Fortunately, there are cases for which the recursive Bayesian estimation problem can be written and solved in closed form or where there are appropriate approximations. In the following subsections, a brief overview about some of these algorithms is given.

4.2.2. Discrete Bayes Filter

If the state space is discrete and finite, the recursive update rule from (4.3) can be rewritten in a discrete version, where the integral is replaced by a sum, see (4.4). This filter is usually called the *discrete* Bayes filter.

$$\text{belief}(x_t) = \eta\, p(z_t \,|\, x_t) \sum_{x_{t-1}} p(x_t \,|\, u_t, x_{t-1})\, \text{belief}(x_{t-1}) \tag{4.4}$$

The implementation of this filter is quite straightforward. For each element x_t of the state space, the equation is evaluated, recursively forming the new belief. A pseudo code implementation of the discrete Bayes filter which also computes the normalizer value η is written down in algorithm 4.1. The relevance of this filter in practice is small, as it enumerates the state space, resulting in a complexity of $O(|X|^2)$ (with $|X|$ being the cardinality of the state space) which makes it computationally unfeasible for larger state spaces. Please note, that in the algorithm the symbols x_t and x_{t-1} as well as the symbols $\text{belief}(x_t)$ and $\text{belief}(x_{t-1})$ have to be regarded independently, just as in (4.4).

4.2.3. Kalman Filter

By the number of different applications in robotics and technology in general, the Kalman filter is a very important Bayesian filter. Therefore, even if it does not play

Algorithm 4.1: Implementation of the discrete recursive Bayes filter for arbitrary, enumerable and finite state spaces, pseudo code originally created for [Arndt 11a].

Input: Previous belief: belief(x_{t-1}), current sensor readings z_t, current control values u_t, motion model $p(x_t \mid u_t, x_{t-1})$, sensor model $p(z_t \mid x_t)$
Output: Current belief: belief(x_t)

```
1: overall_sum ← 0 {keep track of the overall sum to normalize later}
2: for all x_t do
3:     sum ← 0 {calculate the inner sum}
4:     for all x_{t-1} do
5:         sum ← sum + p(x_t | u_t, x_{t-1}) · belief(x_{t-1})
6:     end for
7:     belief(x_t) ← sum · p(z_t | x_t) {calculate the new belief}
8:     overall_sum ← overall_sum + belief'(x_t)
9: end for
10: η ← 1 {the normalizer constant}
11: if overall_sum > 0 then
12:     η ← 1/overall_sum
13: end if
14: for all x_t do
15:     belief(x_t) ← belief(x_t) · η {normalize with η}
16: end for
17: return belief(x_t)
```

a further role in this thesis, it shall still be briefly mentioned. In contrast to the previously introduced discrete Bayes filter, this filter is applicable for continuous state spaces, but is limited to representing Gaussian distributions (that are characterized by their mean μ and their covariance Σ). Another requirement for the classic Kalman filter is that the models (for motion and sensors) need to be linear, to preserve the Gaussian shape of the beliefs. The extended Kalman Filter (EKF) presents a mean for linearizing models that are non-linear.

In the following, the Kalman filtering process as given by [Thrun 05] is replicated, although there exist a lot of other, equivalent notations. Using the previous belief $(\mu_{t-1}, \Sigma_{t-1})$, the current control vector u_t and the current sense vector z_t, the Kalman filter first predicts the a-priori belief $(\overline{\mu}_t, \overline{\Sigma}_t)$ at time t. This is given in (4.5) and (4.6):

$$\overline{\mu}_t = A_t \, \mu_{t-1} + B_t \, u_t \tag{4.5}$$

$$\overline{\Sigma}_t = A_t \, \Sigma_{t-1} \, A_t^T + R_t \tag{4.6}$$

After this step, the update step is applied, which incorporates the new sensor values z_t. This is given in (4.7) to (4.9):

$$K_t = \overline{\Sigma}_t \, C_t^T \, (C_t \, \overline{\Sigma}_t \, C_t^T + Q_t)^{-1} \tag{4.7}$$

$$\mu_t = \overline{\mu}_t + K_t \, (z_t - C_t \, \overline{\mu}_t) \tag{4.8}$$

$$\Sigma_t = (I - K_t \, C_t) \, \overline{\Sigma}_t \tag{4.9}$$

In these equations, A_t and B_t are matrices that describe how the motion model affects the mean and covariance of the belief. R_t describes the covariance of Gaussian noise that is added to the motion model. The matrix C_t represents the sensor model, together with Q_t which describes Gaussian noise of the sensor model, similar as R_t does for the motion model. I denotes the identity matrix. K_t is an intermediate result which is called the Kalman gain.

4.2.4. Particle Filter

Particle filters are different from the previously mentioned filters in the way they represent the current belief. Instead of explicitly storing all the elements of the state space (as in the discrete Bayes filter) or representing them with Gaussians (as in the Kalman Filter), the particle filter represents the current belief using a set of weighted particles. Each particle can be represented by a tuple containing an element x of the state space and a real number w for the weight: $(x, w) \in (X \times \mathbb{R})$. The weight defines how much the respective particle contributes to the overall belief. Usually, the weights of all particles sum up to 1, so the set of all particles represents a probability density function.

The particle filter is a Monte Carlo algorithm[4] which can of course only approximate the correct solution of the Bayesian recursive estimation. On the other hand, there are virtually no limitations regarding the shape of the belief (as the set of particles can approximate any shape[5]) and the models. This is in contrast to e.g. the previously mentioned Kalman filter, which (in its non-extended form) requires both the motion as well as the sensor model to be linear. The particle filters are sometimes also referred to as *Sequential Monte Carlo (SMC)* methods [Arulampalam 02].

The implementation of a basic particle filter is fairly simple. One in pseudo code is given in algorithm 4.2. Similar as the discrete Bayes filter from algorithm 4.1, this one also computes the current belief, belief(x_t) from the previous belief, belief(x_{t-1}) using sensor readings, control values and the models for motion and sensors. The loop starting in line 2 iterates over all the current particles and first applies the motion model (line 3) to generate a new state x_t and then the sensor model (line 4) to compute the new weight. Instead of adding this new particle to the set which is returned, it is added to the set P_t'. This is necessary as the set P_t' first needs to be

[4] Monte Carlo algorithms make use of randomization to generate (not necessarily correct) non-deterministic results.

[5] Depending on the number of particles, the approximation may of course be very rough.

resampled to form the set P_t in line 7. The step of resampling is quite important in the particle filter. There are plenty of methods that have been proposed for resampling. Without going too much into the detail about resampling, its task is to draw particles from P'_t with a probability that is proportional to their weights. This has the goal to duplicate "important" particles (as they have high weights) and to remove particles from the set that only have a low weight. If the number of particles in the filter is not fixed, this step can also be used to increase or decrease the particle count.

Algorithm 4.2: Pseudo code implementation of a particle filter.

Input: Set of weighted particles $P_{t-1} \subset (X \times \mathbb{R})$ representing belief(x_{t-1}), current sensor readings z_t, current control values u_t, motion model $p(x_t \mid u_t, x_{t-1})$, sensor model $p(z_t \mid x_t)$

Output: Set of particles representing belief(x_t)

1: $P_t \leftarrow \emptyset$, $P'_t \leftarrow \emptyset$
2: **for** $(x_{t-1}, w_{t-1}) \in P_{t-1}$ **do**
3: sample $x_t \sim p(x_t \mid u_t, x_{t-1})$ {*draw a sample of the new state according to the motion model*}
4: $w_t = p(z_t \mid x_t)$ {*calculate the new weight according to the sensor model*}
5: $P'_t \leftarrow P'_t \cup (x_t, w_t)$ {*add the new particle to the temporary set*}
6: **end for**
7: Resample N particles from P'_t into P_t
8: **return** P_t

4.3. Graph-Based State Space

As already briefly stated in the introduction, the state space describes all possible values that a state variable can attain. It has already been noted that, in the context of recursive Bayesian estimation, different Bayesian filters put different restrictions on the definition of the state space. For example, the discrete Bayes filter does only work with a discrete and finite state space. More importantly, the definition of the state space is very application-specific and might also depend on the type of sensors and models, that are used.

For target tracking applications using camera images, it might for example be interesting to have a state space that consists of the Cartesian position $\boldsymbol{x}$ and the velocity vector $\dot{\boldsymbol{x}}$ of the target. In other applications, the velocity of the targets might not be observable, in such cases, one might consider removing $\dot{\boldsymbol{x}}$ from the state space and only keeping track of the position.

State spaces are of course not limited to Cartesian spaces. In some applications, it might be wise to use e.g. polar coordinates (maybe in radar tracking). In this thesis, none of these aforementioned state spaces are used, but instead, it is modeled as a graph on which people can move through the environment. This approach offers several advantages, which are explained in detail in the next subsection.

4.3.1. Modeling and Implementation

Regarding the estimation of the position of people in typical environments, an important aspect needs to be noticed: Environments typically do not allow unconstrained movement of people. Usually, indoor environments contain obstacles that cannot be overcome by people, such as walls, furniture and more. In outdoor environments, there might be walls, buildings, plants or fences. These obstacles constitute constraints for human movements that must be modeled in the motion models, to take into account that people do not move over or through these obstacles. But even if an environment theoretically allows unconstrained movement, e.g. if the environment is a large open area, people tend to exhibit typical paths on which they move (see e.g. [Foka 02], [Bennewitz 05]).

Now, instead of handling all these issues in the motion model, another way to cope with them, is to implicitly forbid such motion by creating a state space that does not even allow such transitions. One way of modeling such state spaces is using directed or undirected graphs. If the vertices of a graph have Cartesian positions attached to them, then the edges that connect vertices describe the possible Cartesian positions along which people may move and on which state estimation can be performed (cf. topological maps, discussed in section 2.1.1.2). This idea is not new, but dates back to at least the year 2003 when Liao et al. [Liao 03] published their work about Voronoi Tracking. The basic idea has since then been taken up by several people e.g. [Schulz 03, Schindler 06].

Formally, the graphs and the state space used in the rest of this thesis are defined similarly as by [Liao 03]. The undirected graph $G = (V, E)$ consists of a set of vertices V and a set of edges E. Edge e_{ij} is defined to connect the two vertices v_i and v_j. In this work, both vertices as well as edges can have properties attached. The most important property of a vertex is its Cartesian position in the environment, which is denoted $\boldsymbol{x}_\mathbf{i}$ for vertex v_i. Additionally, a vertex can store transition probabilities $p(e_{\text{outgoing}} \mid e_{\text{incoming}})$ to define with which probability targets take the outgoing edge e_{outgoing} if they come in through edge e_{incoming}. So called door guards can also be attached to vertices. If vertex v_k is guarded by guard d_k, then targets can only traverse through this vertex if the door is in an "open" state. Edges can also have properties attached to them. One example is a probability distribution for the speeds with which targets move along that edge. This could either be a fixed value or e.g. a normal distribution centered around a mean speed. For an illustration of a graph with its properties, see figure 4.2.

Now the underlying structure for the state space has been defined. However, a method to specify a position on that structure (the graph) is still needed to define the state space. The state is defined as a vector (e_{ij}, t) consisting of the edge e_{ij} and a parameter t which is used to linearly interpolate between both ends of the edge. In the extreme case $t = 0$, the position is at the vertex v_i, while in the other extreme, $t = 1$, the position is at the vertex v_j. figure 4.3 shows an example with two different values of t.

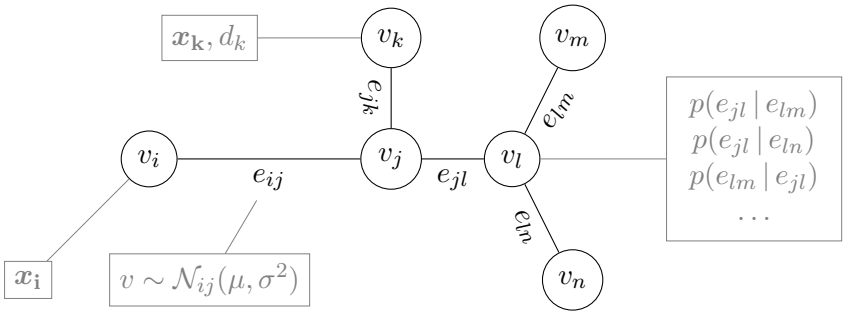


Figure 4.2.: Structure of an example graph with attached properties, originally created for [Arndt 11a] and previously published in [Arndt 12c].

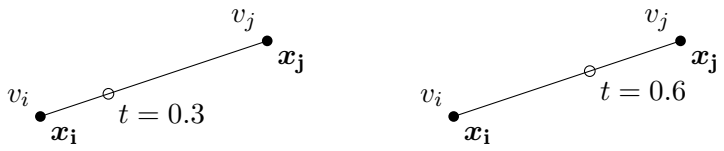


Figure 4.3.: Visualization of the linear interpolation between two vertices v_i, v_j of the graph, originally created for [Arndt 11a].

Formally, the Cartesian position $\boldsymbol{x}$ for a graph-state (e_{ij}, t) can be computed using the linear interpolation as given in (4.10), using the two Cartesian positions of the vertices v_i and v_j.

$$\boldsymbol{x}(e_{ij}, t) = \boldsymbol{x_i} + t \cdot (\boldsymbol{x_j} - \boldsymbol{x_i}) \tag{4.10}$$

4.3.2. Visualization and Mapping to a Grid

With the possibility to compute the Cartesian positions of elements of the state space, *particles* (as used by e.g. a particle filter) can also be easily visualized in a Cartesian representation of the graph. They could e.g. be drawn as circles, similar as done for the two example positions in figure 4.3. By varying the diameter or the color of the circles, it is also possible to visualize different weights of particles. This representation of the belief is elegant and efficient. However, for some scenarios, this way of visualization cannot easily show how the belief is distributed. This is especially the case where there is a large number of particles situated within a small area. In such cases it is hard or impossible to visualize *all* the contributing particles.

Another possibility is to sum up all the particles' weights on one edge and then color or label the graph with these weights. This representation is already a bit more usable and more easy to read, but depending on the lengths of the edges of the graph, it can

be very coarse[6]. Instead, most of the visualizations of beliefs in this thesis make use of a representation using a grid structure. Additionally, in robotics, occupancy grids are a popular method to store probabilistic knowledge about the state of the environment (see section 2.1.1.1), so a possibility to map from a graph to a grid structure might be useful for some applications.

A challenge in the mapping process is the aspect that the edges of the graph are one-dimensional objects that have a length, but no width[7]. The edges of the graph represent links between locations in the environment. In reality, these links (i.e. the paths in the environment) are of course two-dimensional, so that objects (i.e. people, robots, ...) are able to move on them. In practice, an edge rather reflects the area occupied by a rectangle which has two sides parallel to the edge with the same length as that of the edge. The interesting aspect is the length of the other two sides (the *width* of the rectangle). This width could either be assumed to be equal for all edges in the graph or it could be modeled as an edge-property which allows the width to be adapted according to the area represented by that edge.

With this width, a state vector no longer presents a zero-dimensional primitive (i.e. a point) in Cartesian space, but a one-dimensional line segment on which the probability is distributed[8]. There are however difficulties at positions where the graph is not continuously differentiable[9], i.e. where there is no smooth transition in curvature when traversing from one edge to another edge. Unfortunately, this is the case at many vertices of a typical graph, as the edges are represented by piecewise linear functions. For the graph given in figure 4.4a, the extension with rectangles that represent the width can be found in figure 4.4b. This illustration shows the difficulty that arises by regarding the edges as separate entities. Instead, smooth transitions such as visualized in figure 4.4c are favorable as people usually walk on smooth curves, rather then abruptly changing direction. Thus the tracking results are better represented using the method in figure 4.4c. Smooth line joins, however are more complex to compute than rectangles and introduce additional geometric shapes (curves) that need to be handled when mapping to a grid structure.

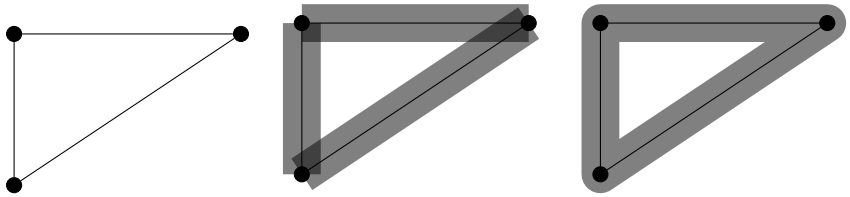
Ignoring varying probabilities on the graph's edges and assuming a uniform distribution of the belief on the graph, methods from the field of computer graphics can be used to map a graph to a grid of arbitrary size and resolution. *Area sampling* [Foley 97, 14.10.2] is a method for mapping geometric structures to a grid (or to pixels). For each grid cell (or pixel), the amount of overlap with the geometric structure is determined

[6] It must be noted that this representation is however highly relevant with respect to safe and cost-efficient mobile robot path planning on a graph, as is shown in detail in section 5.2.

[7] Just like lines or line segments, which are used to graphically represent graph structures.

[8] In this work, the distribution is a simple uniform distribution, but it could be adapted to something more complex.

[9] The term *continuously differentiable* is usually only defined for functions, but it is also well known for parametric curves, such as B-splines, where the property is usually called G^1 *continuity* (see e.g. [Foley 97, 11.2]). If paths on the graph are regarded as parametric curves, the same methods and terminologies can be applied.



(a) Original graph **(b)** Extended with rectangles representing the width **(c)** With smooth line joins

Figure 4.4.: Graph and visualization of edge width in Cartesian space.

and a value proportionally to the overlap is assigned to that cell. For the example graph from figure 4.4c, this process is exemplarily sketched in figure 4.5.

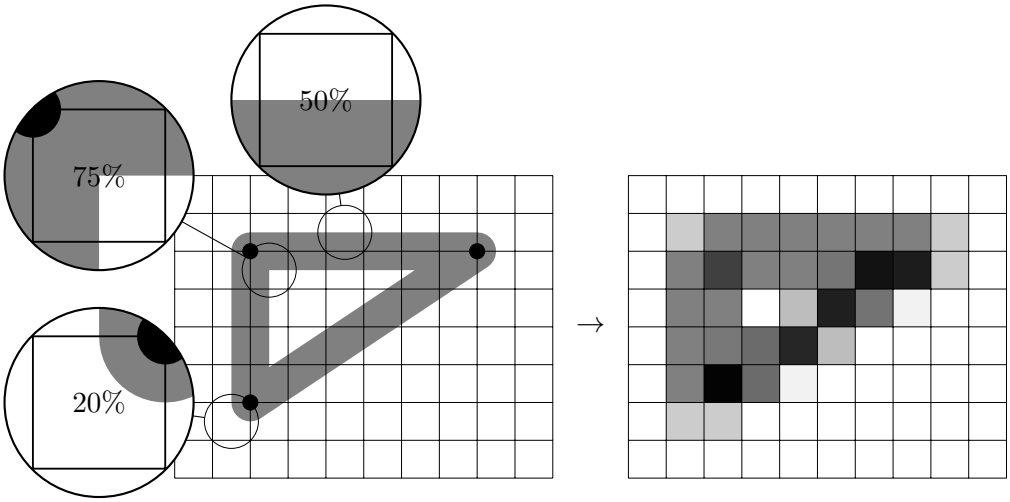


Figure 4.5.: Geometric mapping of graph extended with edge width and smooth line joins onto a grid structure using area sampling.

If the assumption of a uniformly distributed belief is relaxed, the process gets slightly more complicated. Instead of only computing the fractions of overlap between the extended graph and the cells of the grid, these results still need to be multiplied with the integral of the belief along the relevant parts of the graph edges. Unfortunately, exactly this fact makes it hard or even impossible to use readily available library code that implements area sampling. For this reason (and to make use of the fact that the belief is represented by particles), a custom method to map a belief described by particles on a graph to a grid is presented now.

The idea is to simplify the mapping from graph to grid structure using some assumptions. As already stated, the belief is assumed to be approximated by weighted

particles out of which every single one contributes to the overall belief. The original graph has two-dimensional edges that are extended with rectangles as shown in figure 4.4b, i.e. there are *no* smooth line joins. The distribution of the belief along the extended edges in the width-dimension (orthogonal to the edge direction) is assumed to be uniformly distributed. The area of overlap is not calculated exactly, instead everything that is "hit" (even if the amount of overlap is very small) is considered identically. Finally the grid is assumed to have square cells, and all cells are of equal dimension.

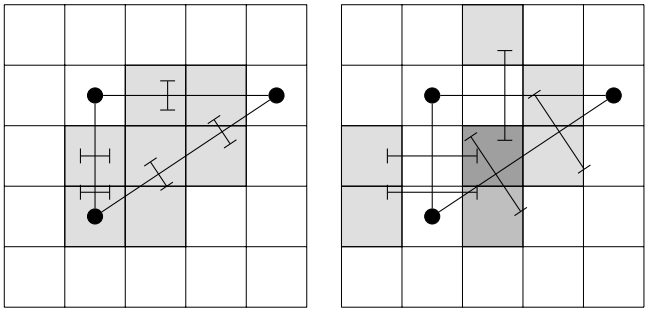
If these prerequisites are satisfied, a schema can be given that defines how to map each particle to a number of grid cells. This number of grid cells is dependent on a number of parameters, e.g. the width of the edge and the step width of the grid. The following schema can be used to calculate all the sample points (positions in the grid) that have to be considered for each particle:

1. Calculate the required numbers of samples on one side of the edge n using (4.11). This inequality is a result of the Nyquist-Shannon theorem and is described in detail including a derivation in section B.1. Its purpose is to compute n large enough so that no cell of the grid is "skipped" during the mapping process.
2. Determine the geometric points where to sample by inserting $i \in [1, 2, \ldots, n]$ into (4.12), once for each side of the edge.
3. For each cell that is "hit" by a sample point, put that cell's identifier in the set of hit particles H.
4. Finally, increment the value of each hit cell $c \in H$ by a value depending on the weighting mode (more on that later), possibly honoring a value limit (of e.g. 1).
5. Optional: Scale the probabilities in all cells to the interval $[0, 1]$ by dividing all probabilities by the maximum probability found in one cell or normalize the values so that they sum up to one.

In the following, these steps are described in more detail. First of all, the number of samples on each side of the edge is an important value that is calculated as a function of the edge width and the step width of the grid. One could argue that sampling exactly twice, at the begin and at the end of a line segment that represents the width of a particle, is enough to map to a grid structure. This is however only true if the width of the edge compared to the step width of the grid is "small"[10]. Please refer to figure 4.6a for a visualization of such a situation. If however the edge width is too large and only two samples are used, the situation in figure 4.6b arises.

In the figure, it can be seen that this situation leads to issues, as cells are missed that should be covered by the particles. The case where the edge width is *equal* to the step width is not depicted, but has similar problems as in many examples for the Nyquist-Shannon sampling rate: In some cases it might work, but in some border cases it fails (if the zero-crossings of the signal are sampled).

[10] In fact, using (4.11), an answer can be given on when this is true. Assume $n = 1$, then (width of the edge $<$ step width of the grid) must hold.



(a) Grid step size 1, edge width 0.5. Everything is fine.

(b) Grid step size 1, edge width 1.5. Bad case, the outer cells are sampled, inner ones may be lost.

Figure 4.6.: Visualization of the application of the Nyquist-Shannon sampling theorem in the context of mapping a belief represented by particles to a grid structure.

Now, with only one sampling point at each side of the particle ($n = 1$), this inequality puts heavy constraints on the grid we may map onto, because it requires rather coarse grids, if the width of the edges is chosen to be large. For this reason, the "sampling frequency" of the process can be increased by dividing the line segment that represents the particle into smaller parts. Here, the required number of samples on one side of the edge n comes into play. By default (and in the examples above) this number equals one ($n = 1$), which means on either side of the graph is one sample (this matches what is depicted in figure 4.6). If however this number n is increased, there are not only two, but $2\,n$ samples in total for a single particle. This increases the sampling frequency and thus decreases the sampling period. Formally writing down the relation between the sampling period (as defined by n and the edge width) and the signal period (as defined by the grid step size) using the Nyquist-Shannon theorem results in (4.11). This formal process can be found in the appendix in section B.1.

$$n > \frac{\text{width of the edge}}{\text{step width of the grid}} \tag{4.11}$$

The same situation as in figure 4.6b, but this time with $n = 2$ is depicted in figure 4.7. As the inequality now holds, the result is correct.

Determining the geometric points of the samples is straightforward by looking at figure 4.8. In this, figure $\boldsymbol{x}_{\text{particle}}$ denotes the Cartesian position of the particle, w_{edge} the width of the edge and $\boldsymbol{n}_{\text{edge}}$ the normal-vector of the edge. Two exemplary sample positions $\boldsymbol{x}_{\text{sample}_1}$ and $\boldsymbol{x}_{\text{sample}_2}$ have been marked in the figure. These sample points can generally be computed by inserting $i \in [1, 2, \ldots, n]$ into (4.12), once for each side

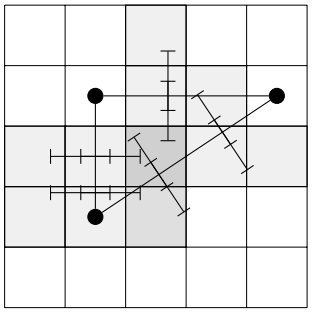


Figure 4.7.: Grid step size 1, edge width 1.5. With $n = 2$, the sampling is correct.

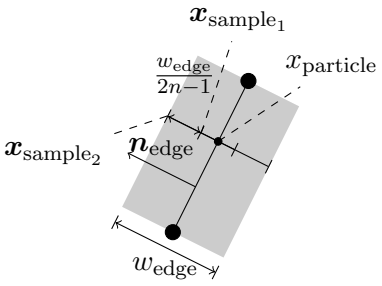


Figure 4.8.: Calculation of the points where to sample for an example edge with $n = 2$

of the edge (note the $\pm$, which indicates that for each i, there are two samples, one on each side of the edge).

$$\boldsymbol{x}_{\text{sample}} = \boldsymbol{x}_{\text{particle}} \pm \boldsymbol{n}_{\text{edge}} \cdot (0.5 + (i-1)) \cdot \frac{w_{\text{edge}}}{2n-1} \tag{4.12}$$

Next, it must be kept track of *which* cell each of the positions $\boldsymbol{x}_{\text{sample}}$ map to. For this reason, the set H keeps track of the cells hit by the current particle. If near the border of the grid, the case may arise that not all positions hit a grid cell. It is also possible that two samples hit the same grid cell. As a set only contains unique elements, such a cell is then only contained once in H.

As soon as all samples of a particle have been analyzed, the cells contained in H (i.e. the ones that have been hit by that particle) need to be modified. Generally, this involves summing a new value onto the already existing value. At this point, saturation arithmetics[11] may be needed so that the value of a single cell does not exceed a given value, e.g. a probability of 1.

[11] With saturation arithmetics it is possible to limit the outcomes of arithmetic operations such as addition, subtraction, multiplication etc. to a given interval. In the context of occupancy grid cells this would be the interval $[0, 1]$.

But which values should be summed to the cell's current value? This depends on the chosen *weighting method*, which defines in which way the value of the particle is mapped to the cells. Two different weighting methods shall be defined for this thesis. The first one is the *full* weighting, which is very simple: For each cell hit by a particle, the *full* weight of the particle (v = probability of particle) is added to each cell. The second weighting method is the *proportional* weighting. The idea of this method is to spread the weight of the particle evenly over all hit cells. In this case, the value is calculated as given in (4.13). The symbol $|H|$ describes the cardinality of the set H, i.e. the number of hit cells.

$$v = \frac{\text{probability of particle}}{|H|} \tag{4.13}$$

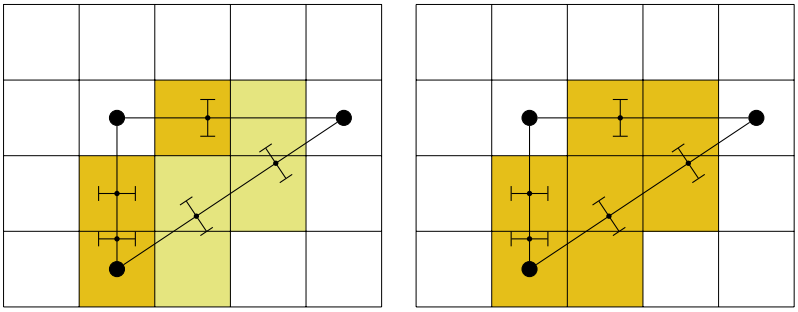
Each of these two weighting methods has its own applications. If for example an occupancy graph (which is described later in section 4.8 and applied even later in section 5.1.5), is mapped to an occupancy grid, a "pessimistic" mapping from graph to grid is needed. Pessimistic means that if e.g. an edge of the graph is blocked by an obstacle, the whole path in a grid representation also needs to be blocked. This cannot be established with proportional mapping, but only with full mapping, as this assigns the full particle probability to the full width of the edge.

In a different, very typical scenario, one might require to map a belief (a probability density function) to a grid structure while maintaining the characteristics of the PDF (an integral of one). In such a case, it is favorable to use the proportional mapping, as it maintains the integral (or in the discrete case, the sum) of one. For visualization purposes, both weighting methods are usable, although they may require normalizing or scaling to ensure that the viewer is able to clearly distinguish between different probabilities/weights.

In figure 4.9, the graph that has been used throughout this section is mapped to a grid using the two different weighting methods and different edge widths. For better distinguishability of different values, in the visualizations, a colormap has been used to encode the different weights.

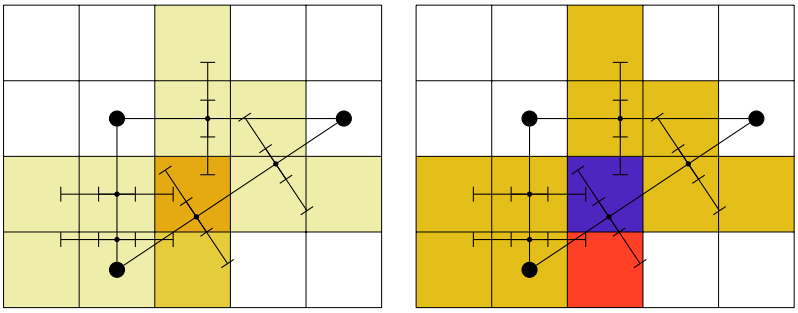
4.4. Brownian Motion on Graphs

Generally, in the field of probabilistic state estimation, the motion model describes how the system evolves over time. In scope of this thesis, the state to be estimated contains the positions of people in the environment. In the applications where the positions of the people are *tracked*, it is a target tracking scenario, in which the motion model describes how targets (i.e. people) behave over time. In target tracking applications, there are usually no control variables that influence the behavior, so these motion models are basically functions of the previous state and the time that



(a) Edge width 0.5, proportional weighting.

(b) Edge width 0.5, full weighting.



(c) Edge width 1.5, proportional weighting.

(d) Edge width 1.5, full weighting.

(e) Colormap

Figure 4.9.: Mapping of five particles with a weight of 0.25 each onto a grid structure using different edge widths and different weighting methods.

has passed. Formally, the motion model gives the probability $p(x_t \mid x_{t-1})$ to be in the new state x_t given the previous state x_{t-1}. The passed time is included in this term through the indices.

Brownian motion usually refers to the random movement of particles in gases or liquids. However, the term has also been used in the tracking context, e.g. by Montemerlo et al. [Montemerlo 02, II. E.], who have used it to model the motions of humans. Brownian motion is a very universal model for motion, as it relies on very few assumptions. For a Cartesian state space, Brownian-like motion can easily be modeled if a direction vector is randomly drawn from e.g. a uniform distribution and a motion with a length drawn from e.g. a normal distribution is applied.

4.4.1. Implementation and Examples

In the scope of this work, the state space is not Cartesian, but represented by a graph (see previous section). In this case, the Brownian motion model also draws a length which describes the distance to be traveled on the graph, but instead of using an arbitrary direction vector, the current edge is traversed in either forward or backward direction. At vertices, the decision must be made which outgoing edge to take. This decision is also based on the transition properties attached to the graph. This process is realized using the procedure **ApplyMotion** which is given in pseudo code in algorithm 4.3. For a current state (e_{ij}, t), the procedure is initially called with the drawn distance to travel, i.e. **ApplyMotion**(e_{ij}, t, d). In line 2, the amount to modify the current parameter t is calculated by dividing the distance to be traveled by the Cartesian length of that edge. Depending on the value of the new value t', the procedure can either stay on the same edge (the easy case, which is handled in line 5) or transition to another edge. In the latter case, a new edge has to be drawn from either the lists of predecessors or the successors of this edge. In these cases, the remaining distance to travel d' is calculated and the procedure calls itself recursively in line 16.

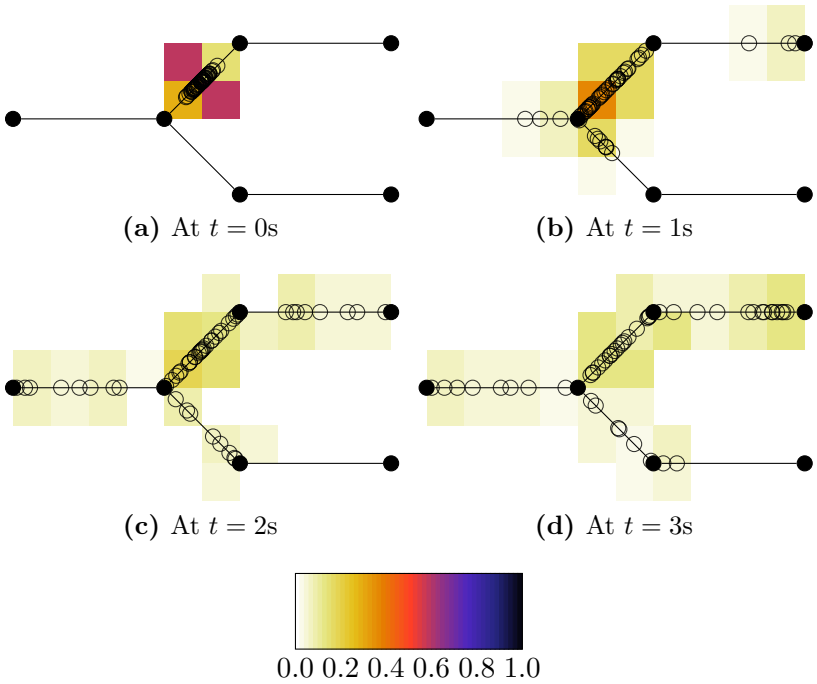
Algorithm 4.3: Traversing the graph randomly by a given distance, algorithm originally created for [Arndt 11a].

Input: Current edge e_{ij}, current value of t and the distance to travel d
Output: The new edge e_{ij} and the new value of t
Require: $\forall i, j \, \|\boldsymbol{x_i} - \boldsymbol{x_j}\| > 0$

1: **ApplyMotion**(e_{ij}, t, d) :
2: $\delta_t \leftarrow d \,/\, \|\boldsymbol{x_i} - \boldsymbol{x_j}\|$
3: $t' \leftarrow t + \delta_t$ {*calculate new t value*}
4: **if** $t' \in [0, 1]$ **then**
5: **return** (e_{ij}, t') {*stay on the same edge, just adjust t*}
6: **else**
7: **if** $t' < 0$ **then**
8: $e_{kl} \sim$ **Pred**(e_{ij}) {*draw new edge e_{jk} from the predecessors of e_{ij}*}
9: $t' \leftarrow 1$
10: $d' \leftarrow d + t \, \|\boldsymbol{x_i} - \boldsymbol{x_j}\|$
11: **else**
12: $e_{kl} \sim$ **Suc**(e_{ij}) {*draw new edge e_{jk} from the successors of e_{ij}*}
13: $t' \leftarrow 0$
14: $d' \leftarrow d - (1 - t) \, \|\boldsymbol{x_i} - \boldsymbol{x_j}\|$
15: **end if**
16: **return** **ApplyMotion**(e_{kl}, t', d')
17: **end if**

This Brownian motion model in action can be visualized by looking at snapshots taken at different points in time. In these snapshots, particle positions are indicated by circles on the graph. Their weights are all equal and are not modified by the motion model. Additionally to the direct visualization of the particles, the belief is

also mapped to a grid structure to better visualize the accumulation of particles. The initial situation at $t = 0$ is shown in figure 4.10a. This situation represents the case where the belief is concentrated in a small area around a position on the graph, i.e. the person is situated at that one position with only small uncertainty. The following subfigures 4.10c to 4.10d represent situations after which the particles have been moved along the edges of the graph, according to a Brownian motion model. The highest probability is still centered near the original position, but the belief is spread out further, the more time has passed.



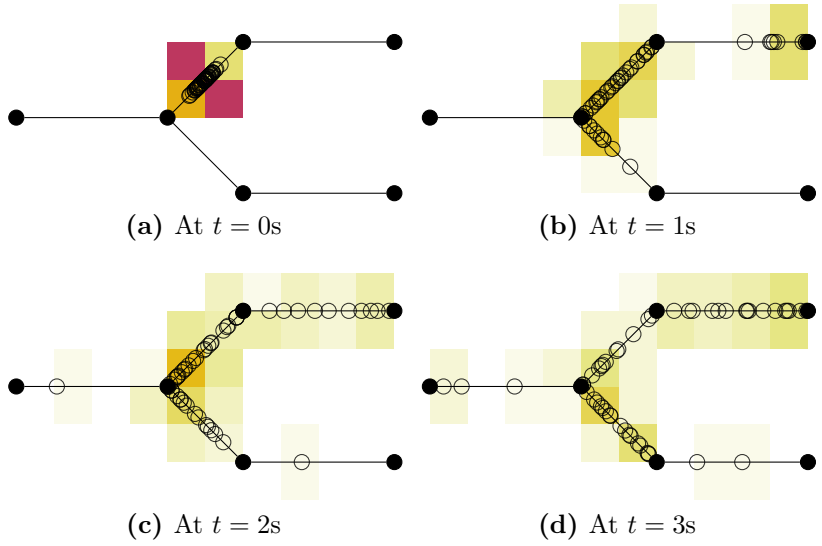
(a) At $t = 0$s **(b)** At $t = 1$s

(c) At $t = 2$s **(d)** At $t = 3$s

(e) Legend for the colors

Figure 4.10.: Visualization of Brownian motion on the tracking graph.

As previously stated (in section 4.3) and implemented in the algorithm, a graph as defined for this thesis, can store non-uniform transition probabilities $p(e_{\text{outgoing}} \mid e_{\text{incoming}})$ for a target passing vertices. In the above example, uniform probabilities have been assumed. If the transition probabilities at the vertex in the center of the graph are changed, so that it is more likely to pass from the upper part of the graph to the lower part of the graph (in contrast to the part at the left), then the situation after applying the motion model looks different, as can be seen in figure 4.11. In this case, the probability for traversing "top-down" has been set to 0.8, so that the probabilities of traversing "top-top" and "top-left" are set to 0.1 each. The change in particle distribution can easily be seen by comparing the two figures.



(a) At $t = 0$s **(b)** At $t = 1$s

(c) At $t = 2$s **(d)** At $t = 3$s

Figure 4.11.: Visualization of Brownian motion on the tracking graph with non-uniform edge-transition probabilities.

4.5. Sensor Model for PIR Sensors on Wireless Sensor Nodes

A sensor model typically takes the uncertainties and failures associated with the sensor it represents into account. The classic (forward) sensor model describes the probability of measuring something, given a state of the environment. Formally, it gives the probability $p(z \mid x)$ to observe the event z given the current state x. For some applications, the *marginalized inverse* sensor model $p(x_i \mid z)$ is needed, as can be seen later.

Usually, each sensor requires its own sensor model. In robotics, there are many examples for sensor models of many different sensors, e.g. laser rangefinders [Thrun 00, Fox 03], ultrasonic sensors [Elfes 87] and many more. For this work, the main sensor type is the passive infrared sensor (PIR) which is mounted on the wireless sensor nodes (as described in detail in section A.1). In this section, the characteristics of this sensor are examined and a sensor model is developed.

Passive infrared sensors are passive sensors (i.e. they do not emit energy for the sensing process), that register slight changes in temperature by measuring and comparing levels of infrared radiation. This radiation is situated at the far infrared spectrum, which still behaves similar as near infrared or visible light. Because PIR sensors can only register *changes* in temperature, they are naturally *motion* detectors. Only objects that are moving and additionally cause a large enough change in the perceived infrared radiation can be detected by the measurement principle. A typical example of objects that can be easily detected using such sensors are moving humans or (warm-blooded) animals. PIR sensors have lenses with defined opening angles in

horizontal and vertical direction and a limited range in which they are able to reliably detect motion.

In a typical scenario, several of these PIR sensors are mounted at different positions in the environment. These mounting positions need to be defined in the sensor model, see figure 4.12 for a visualization of most of the relevant parameters. It shows the mounting height h_m, the sensor range r_s and the mounting angles α, β, γ. The two-dimensional (the third coordinate is contained in h_m) mounting position $(x, y)^T$ is also shown. The vertical and horizontal opening angles α_v and α_h are not visualized.

In the following, for simplicity, the values of x, y, α, β and γ are assumed to be 0, unless noted otherwise. Thus, all points are regarded in *sensor coordinates*. The transformation between sensor and *global* coordinates is later established using rigid transformation matrices.

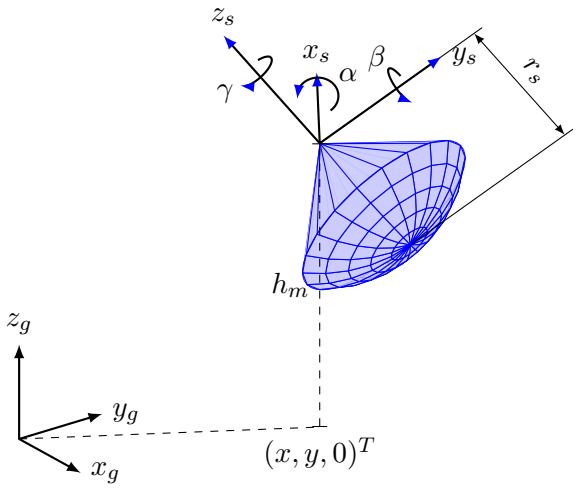


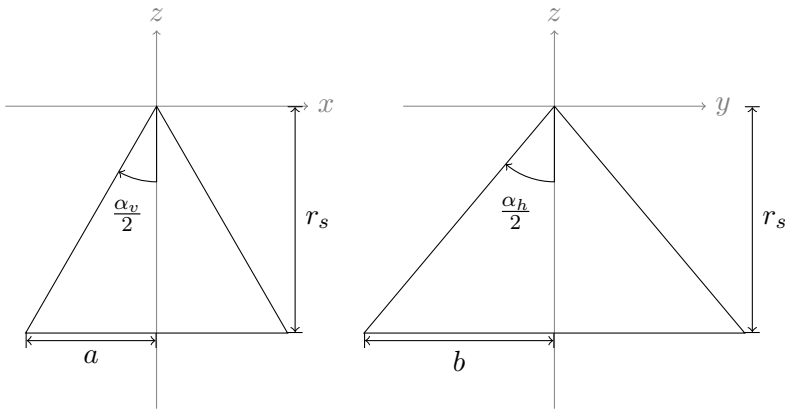
Figure 4.12.: Schematic view of the sensing volume of a PIR sensor and the mounting pose relative to the global coordinate system. In the examples, the angles are $\alpha = 15°$, $\beta = -60°$ and $\gamma = -10°$.

In previous works, starting with [Arndt 11a], the two sensor planes (x-y and y-z) have been regarded separately and independently from each other. This assumption of independence was very strong and the resulting sensor model was thus only a rough model of reality. It had however the advantage of being computationally simple and easy to understand. For this thesis, a more complex, but also much more realistic sensor model was designed.

This new sensor model uses the real three-dimensional description of the sensing volume. First of all, a geometric description of the sensing volume is needed. At first glimpse, the sensing volume appears to be an elliptic cone[12], which is however not true. The elliptic cone *does* account for the two opening angles (one horizontal, one

[12] I.e. a cone with an elliptical base, in contrast to a circular one.

vertical) which are usually not equal. However, it does *not* properly model the limited range of the sensor. The range of the sensor r_s limits the volume of the elliptic cone to all those points that *also* lie within a sphere with radius r_s, centered at the sensor mounting position. Geometrically combining these two insights, the sensing volume can be represented as the intersection of an elliptic cone and a sphere. The exact parameters a and b of the elliptic cone can be computed using the two opening angles α_v, α_h and the sensor range r_s using the sketches in figure 4.13.



(a) Projected onto the x-z plane.

(b) Projected onto the y-z plane.

Figure 4.13.: Calculation of the parameters of the elliptic cone representing part of the sensor volume.

One important aspect must be noted first: The elliptic cone has its apex at the point $(0, 0, 0)^T$ and the cone is facing downwards the z-axis (direction $(0, 0, -1)^T$). This assumption makes the following calculations easier and does not violate the generality of the solutions, as for other mounting poses (positions as well as rotations), all coordinates can be transformed from and to this view on the cone.

The height of the elliptic cone is equal to the sensor range $h = r_s$. It cannot be smaller, because otherwise, the point "straight" down the z-axis are not properly included. The parameter a (the major radius of the base ellipse, along the x-axis) can be calculated as:

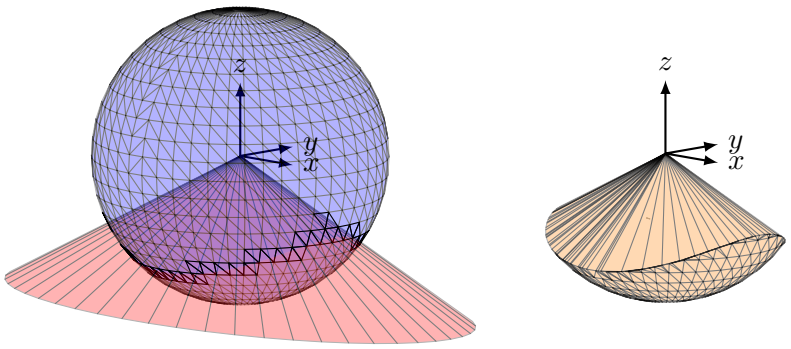
$$\tan\left(\frac{\alpha_v}{2}\right) = \frac{a}{r_s} \tag{4.14}$$

$$r_s \cdot \tan\left(\frac{\alpha_v}{2}\right) = a \tag{4.15}$$

Analogously, the parameter b (the minor radius of the base ellipse, along the y-axis) can be calculated as:

$$r_s \cdot \tan\left(\frac{\alpha_h}{2}\right) = b \tag{4.16}$$

The sphere is more simple. As already mentioned, it is regarded in sensor coordinates at the origin $(0, 0, 0)^T$ with a radius of r_s. The final sensor volume is the intersection of both the elliptic cone and the sphere. The intersection process is visualized in figure 4.14a where the original objects are drawn transparently in the same graphic. The resulting shape looks like rendered in figure 4.14b.



(a) Intersection of the geometric entities.

(b) The resulting sensor volume itself.

Figure 4.14.: Visualization of the 3D sensor volume of the PIR sensors.

With the sensor volume being geometrically described, two tasks need to be solved: First, there must be a method to check if a point is included in the sensor volume. Second, a method to intersect a line with the sensor volume is needed to calculate the amount of partial overlaps of the sensor volume with targets. Please note, that the sensor is still assumed to be mounted at $(0, 0, 0)^T$, facing downwards the z-axis. This assumption is relaxed later.

Checking if a point (in sensor coordinates) is contained in the sensing volume is done in two steps. First, the length of the vector $\boldsymbol{p} = (p_x, p_y, p_z)^T$ of the point is compared with the sensor range r_s. If the length is larger than the sensor range, it cannot be contained in the volume (this checks the "sphere" part of the intersection). This gives the first condition that must be fulfilled:

$$\|\boldsymbol{p}\| \leq r_s \tag{4.17}$$

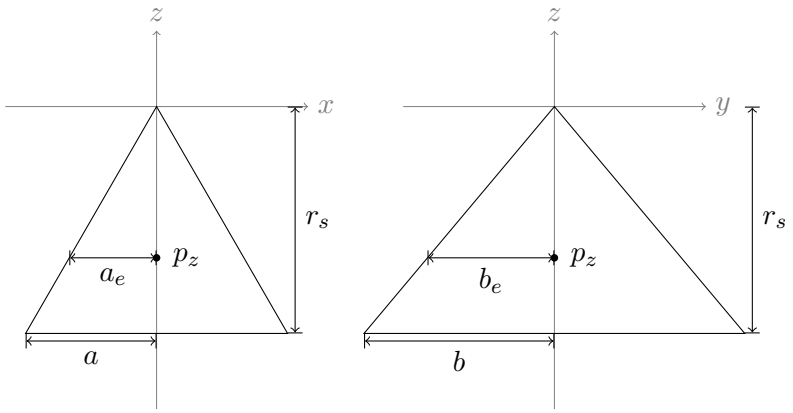
$$\sqrt{p_x^2 + p_y^2 + p_z^2} \leq r_s \tag{4.18}$$

The second condition checks for the inclusion of the point in the elliptic cone. Checking if a point $\boldsymbol{p} = (p_x, p_y, p_z)^T$ is contained in the elliptic cone can be broken down into extracting an elliptical cross-section of the elliptic cone at the height p_z and checking if the 2D-coordinate $(p_x, p_y)^T$ lies within the (2D) ellipse.

The radii a_e and b_e of the ellipse can be computed by linearly interpolating between the apex and the base of the elliptic cone with $p_z/(-r_s)$ being the factor of the interpolation (see the visualization in figure 4.15), resulting in:

$$a_e = a \cdot \frac{p_z}{-r_s} \tag{4.19}$$

$$b_e = b \cdot \frac{p_z}{-r_s} \tag{4.20}$$



(a) Projected onto the x-z plane.

(b) Projected onto the y-z plane.

Figure 4.15.: Calculation of the parameters of the elliptic cross-section of the elliptic cone for the z-position p_z.

Using the Cartesian equation of an ellipse, centered at $(0, 0)^T$, the check for inclusion of the point in the elliptical cross-section of the elliptic cone can be written as:

$$\left(\frac{p_x}{a_e}\right)^2 + \left(\frac{p_y}{b_e}\right)^2 \leq 1 \tag{4.21}$$

There is another condition necessary to fully define the second condition (the inclusion of the point in the elliptic cone). The elliptic cone needs to be limited to negative z-values. Mathematically, positive z-values are possible, but they lie "behind" the sensor, and must thus not be considered, so additionally the following must hold:

$$p_z \leq 0 \tag{4.22}$$

In total, when combining these three conditions, the following predicate is true if and only if the point $\boldsymbol{p}$ (in sensor coordinates) lies within the sensor volume (which is parameterized by the range r_s and the radii a_e and b_e of the elliptic cone):

$$\textbf{InVolume}(\boldsymbol{p}) = (\|\boldsymbol{p}\| \leq r_s) \wedge (p_z \leq 0) \wedge \left(\left(\frac{p_x}{a_e}\right)^2 + \left(\frac{p_y}{b_e}\right)^2 \leq 1 \right) \tag{4.23}$$

4.5.1. Calculation of the Confidence

The actual sensor model is based on the amount of overlap of a (model) target with the sensor volume. If the whole target height h_t lies in the sensing range, the confidence for detecting the target is defined as 1. If the sensor cannot detect the whole target (in terms of its height, projected into the sensing range), the confidence for detecting the target is smaller than one and is equal to the fraction of h_t which can be observed. Consequently, if the target cannot be observed at all, the confidence value is 0.

To calculate the overlap, the target is assumed to have the shape of a line segment starting at the two-dimensional (global) position (x, y) of the target at floor level and reaching to the same two-dimensional position, but at height h_t (the height a target is assumed to have to "perfectly" trigger the sensor). The lower ($\boldsymbol{x}_t^l$) and the upper ($\boldsymbol{x}_t^u$) boundaries of the target are calculated according to (4.24) and (4.25). Here, the two-dimensional target position $(x, y)^T$ is assumed to be in *global* coordinates, beginning at the floor $z = 0$ and reaching to the target height $z = h_t$. These coordinates are transformed from global to sensor coordinates using the inverse transformation matrix M' which is explained in detail later.

$$\boldsymbol{x}_t^l = M' \cdot (x, y, 0)^T \tag{4.24}$$

$$\boldsymbol{x}_t^u = M' \cdot (x, y, h_t)^T \tag{4.25}$$

The overlap is calculated as follows: If both points (start and end of the line segment) lie within the sensor volume (as determined by the predicate in (4.23) (applied to the transformed coordinates), then there is full overlap (by the convex property of the volume) and the confidence is assigned a value of 1.

If one of the points does not lie in the sensing volume, a ray[13] originating from the lower point $\boldsymbol{x}_t^l$ must be intersected in "up direction" (mathematically $(\boldsymbol{x}_t^u - \boldsymbol{x}_t^l)$)

[13] A ray (also called a half-line) is originating from one point and then proceeding straight like a line in its direction vector.

with the volume and the upper one ($\boldsymbol{x}_t^u$) must be intersected in "down direction" (mathematically ($\boldsymbol{x}_t^l - \boldsymbol{x}_t^u$)). The line segment between these two intersections then represents the overlap. If there is no intersection, the overlap is zero. In the following, this process is given more formally.

The relevant length of overlap l_r can be calculated using algorithm 4.4. First, the function **IntersectSensorVolumeWithRay** is called twice, once for the ray originating from the lower point and once for the ray originating from the upper point. The result of this function is a tuple that contains information whether an intersection was found in either direction (stored in $i_\uparrow$ and $i_\downarrow$) and the solution of the intersection process, i.e. the point of intersection (stored in $\boldsymbol{s}_\uparrow$ and $\boldsymbol{s}_\downarrow$). Using this information, the relevant length l_r is then calculated. The function **IntersectSensorVolumeWithRay** itself is given in algorithm 4.5. It takes the base vector $\boldsymbol{p}$ and the direction vector $\boldsymbol{r}$ of the ray as arguments.

Algorithm 4.4: Calculation of the relevant length l_r of the overlap.

1: $(i_\uparrow, \boldsymbol{s}_\uparrow) \leftarrow$ **IntersectSensorVolumeWithRay**$(\boldsymbol{x}_t^l, \boldsymbol{x}_t^u - \boldsymbol{x}_t^l)$ {*For the direction up, determine if ($i_\uparrow$) and where ($\boldsymbol{s}_\uparrow$) there is an intersection.*}

2: $(i_\downarrow, \boldsymbol{s}_\downarrow) \leftarrow$ **IntersectSensorVolumeWithRay**$(\boldsymbol{x}_t^u, \boldsymbol{x}_t^l - \boldsymbol{x}_t^u)$ {*For the direction down, determine if ($i_\downarrow$) and where ($\boldsymbol{s}_\downarrow$) there is an intersection.*}

3: $$l_r = \begin{cases} h_t & \textbf{InVolume}(\boldsymbol{x}_t^l) \wedge \textbf{InVolume}(\boldsymbol{x}_t^u) \\ \|\boldsymbol{x}_t^l - \boldsymbol{s}_\uparrow\| & \textbf{InVolume}(\boldsymbol{x}_t^l) \wedge i_\uparrow \\ \|\boldsymbol{x}_t^u - \boldsymbol{s}_\downarrow\| & \textbf{InVolume}(\boldsymbol{x}_t^u) \wedge i_\downarrow \\ 0 & \text{else} \end{cases}$$

Similarly as for the checks of inclusion in the sensor volume, also the intersection (of a ray) with the sensor volume needs to be split into two intersections. One for the sphere-part and one for the elliptic-cone-part. The functions performing these checks are called **IntersectEllipticConeWithRay** and **IntersectSphereWithRay**. However, having obtained a point of intersection does not yet guarantee that this point also lies within the sensing volume. If there is a positive intersection with the sphere, it may be that this point does still not lie withing the bounds of the elliptic cone. If it does, everything is fine and this point is the final intersection point. If not, the intersection with the elliptic cone must be checked. If it lies within the sensing volume, this is the final intersection point. If not, the point does not lie within the bounds of the sphere and thus does not qualify. In that case, there is no intersection at all.

The functions **IntersectEllipticConeWithRay** and **IntersectSphereWithRay** are given in algorithm 4.6 and algorithm 4.7, respectively. To define these functions, the intersection points of a generic line with a sphere and an elliptic cone need to be calculated. This can be accomplished by solving quadratic equations. For brevity, these calculations are not given here, but in the appendix in section B.8. and section B.9.

For both the elliptic cone as well as the sphere, the values t_1 and t_2 are the intermediate results. If one of these t_i is negative, it can be discarded right away (as this indicates

Algorithm 4.5: Algorithm to intersect the sensor volume with a ray.

Parameter: Vertical opening angle of the sensor $\alpha_v \in \mathbb{R}$
Parameter: Horizontal opening angle of the sensor $\alpha_h \in \mathbb{R}$
Parameter: Sensor range of the sensor $r_s \in \mathbb{R}$
Input: Start of the ray $\boldsymbol{p} \in \mathbb{R}^3$
Input: Direction of the ray $\boldsymbol{r} \in \mathbb{R}^3$
Output: Pair $\mathbb{B} \times \mathbb{R}^3$ denoting if an intersection was found and where the intersection point lies

1: **IntersectSensorVolumeWithRay**$(\boldsymbol{p}, \boldsymbol{r})$
2: $a \leftarrow r_s \cdot \sin(\alpha_v/2)$ {*Calculate major radius of base of elliptic cone.*}
3: $b \leftarrow r_s \cdot \sin(\alpha_h/2)$ {*Calculate the minor radius of base of elliptic cone.*}
4: $(i_c, \boldsymbol{s}_c) \leftarrow$ **IntersectEllipticConeWithRay**$(a, b, r_s, \boldsymbol{p}, \boldsymbol{r})$
5: $(i_s, \boldsymbol{s}_s) \leftarrow$ **IntersectSphereWithRay**$(r_s, \boldsymbol{p}, \boldsymbol{r})$
6: **if** $i_s = 1$ **then**
7: {*There is an intersection with the sphere, but it may be that the solution is no longer within the elliptic cone bounds. Check that here.*}
8: **if InVolume**$(\boldsymbol{s}_s)$ **then**
9: **return** $(1, \boldsymbol{s}_s)$ {*In bounds, use this as the intersection.*}
10: **end if**
11: **end if**
12: **if** $i_c = 1$ **then**
13: {*There is an intersection with the elliptic cone, but it may be that the solution is no longer within the sphere bounds. Check that here.*}
14: **if InVolume**$(\boldsymbol{s}_c)$ **then**
15: **return** $(1, \boldsymbol{s}_c)$ {*In bounds, use this as the intersection.*}
16: **end if**
17: **end if**
18: **return** $(0, \text{undef})$ {*No intersections found or none of the found intersections were in bounds, return "no intersection".*}

that the resulting intersection would lie on the wrong "side" of the ray. The remaining t is then inserted into the line equation and the resulting point is checked for validity.

Calculating the overlap and thus the confidence is then as simple as calculating the ratio between the relevant length (as given by algorithm 4.4) and the model target height:

$$\textbf{Confidence}(x, y) = \frac{l_r}{h_t} \tag{4.26}$$

4.5.2. Transformation between Global and Sensor Coordinates

As already stated earlier, coordinates so far have been assumed to be in sensor coordinates, relative to the sensor mounting pose $(0, 0, 0, 0°, 0°, 0°)^T$. To be able to use world coordinates, it is necessary to transform between global and local coordinates.

Algorithm 4.6: Algorithm to intersect the elliptic cone with a ray.

Input: Major radius of the base of the elliptic cone $a \in \mathbb{R}$
Input: Minor radius of the base of the elliptic cone $b \in \mathbb{R}$
Input: Height of the elliptic cone $h \in \mathbb{R}$
Input: Start of the ray $\boldsymbol{p} \in \mathbb{R}^3$
Input: Direction of the ray $\boldsymbol{r} in \mathbb{R}^3$
Output: Pair $\mathbb{B} \times \mathbb{R}^3$ denoting if an intersection was found and where the intersection point lies

IntersectEllipticConeWithRay$(a, b, h, \boldsymbol{p}, \boldsymbol{r})$
$q_a = (h^2 \cdot r_x^2)/a^2 + (h^2 \cdot r_y^2)/b^2 - r_z^2$
$q_b = (2p_x \cdot r_x \cdot h^2)/a^2 + (2p_y \cdot r_y \cdot h^2)/b^2 - 2p_z \cdot r_z$
$q_c = (h^2 \cdot p_x^2)/a^2 + (h^2 \cdot p_y^2)/b^2 - p_z^2$
if $q_b^2 - 4q_a \cdot q_c < 0$ **then**
    return $(0, \text{undef})$ {*No solution in this case*}
end if
$t_{1,2} = (-q_b \pm \sqrt{q_b^2 - 4 \cdot q_a \cdot q_c})/(2 \cdot q_a)$
$t \leftarrow 0$ {*Choose the smaller, non-negative t to form the solution* $\boldsymbol{s}$}
if $t_1 > 0$ **then**
    $t \leftarrow t_1$
end if
if $t_2 > 0 \wedge t_2 < t_1$ **then**
    $t \leftarrow t_2$
end if
$\boldsymbol{s} = \boldsymbol{p} + t \cdot \boldsymbol{r}$
if $s_z > 0$ **then**
    return $(0, \text{undef})$ {*Would hit the upper half of the cone above zero, not valid.*}
end if
if $s_z < -h$ **then**
    return $(0, \text{undef})$ {*Would hit the (infinite) cone where it is not valid anymore.*}
end if
return $(1, \boldsymbol{s})$

This coordinate transformation is accomplished using homogeneous rotation and transformation matrices. The following rotation matrices and the following transformation matrix are used (the trigonometric functions $\sin(x)$ and $\cos(x)$ are abbreviated by $s(x)$ and $c(x)$, respectively):

- $R_x(\alpha)$ rotates around the x axis with the roll angle α, see (4.27)
- $R_y(\beta)$ rotates around the y axis with the pitch angle β, see (4.28)
- $R_z(\gamma)$ rotates around the z axis with the yaw angle γ, see (4.29)
- $T(x, y, z)$ translates along the three axis, see (4.30)

Algorithm 4.7: Algorithm to intersect the sphere with a ray.

Input: Radius of the sphere $r \in \mathbb{R}$
Input: Start of the ray $\boldsymbol{p} \in \mathbb{R}^3$
Input: Direction of the ray $\boldsymbol{r} \in \mathbb{R}^3$
Output: Pair $\mathbb{B} \times \mathbb{R}^3$ denoting if an intersection was found and where the intersection point lies

IntersectSphereWithRay$(r, \boldsymbol{p}, \boldsymbol{r})$
$q_a = r_x^2 + r_y^2 + r_z^2$
$q_b = 2p_x \cdot r_x + 2p_y \cdot r_y + 2 \cdot p_z \cdot r_z$
$q_c = p_x^2 + p_y^2 + p_z^2 - r^2$
if $q_b^2 - 4q_a \cdot q_c < 0$ **then**
  return $(0, \text{undef})$ {*No solution in this case*}
end if
$t_{1,2} = (-q_b \pm \sqrt{q_b^2 - 4 \cdot q_a \cdot q_c})/(2 \cdot q_a)$
if $t_1 < 0 \wedge t_2 < 0$ **then**
  return $(0, \text{undef})$ {*Both solutions are in the wrong direction of the ray.*}
end if
$t \leftarrow 0$ {*Choose the smaller, non-negative* t *to form the solution* $\boldsymbol{s}$}
if $t_1 > 0$ **then**
  $t \leftarrow t_1$
end if
if $t_2 > 0 \wedge t_2 < t_1$ **then**
  $t \leftarrow t_2$
end if
$\boldsymbol{s} = \boldsymbol{p} + t \cdot \boldsymbol{r}$
if $s_z > 0$ **then**
  return $(0, \text{undef})$ {*Would hit the upper half of the sphere, above zero, not valid.*}
end if
return $(1, \boldsymbol{s})$

$$R_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & c(\alpha) & -s(\alpha) & 0 \\ 0 & s(\alpha) & c(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.27}$$

$$R_y(\beta) = \begin{pmatrix} c(\beta) & 0 & s(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -s(\beta) & 0 & c(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.28}$$

$$R_z(\gamma) = \begin{pmatrix} c(\gamma) & -s(\gamma) & 0 & 0 \\ s(\gamma) & c(\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.29}$$

$$T(x, y, z) = \begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.30}$$

The forward transformation (from sensor coordinates to global coordinates) can be computed using the homogeneous transformation matrix $M(\alpha, \beta, \gamma, x, y, z)$. This matrix rotates using the roll-pitch-yaw mounting angles α, β and γ and translates the coordinates to the real-world mounting position $(x, y, z)^T$ of the sensor. The matrix M has the form given in (4.32). The exact entries and their calculation as a function of the mounting pose are given in the appendix in section B.10.1.

$$M(\alpha, \beta, \gamma, x, y, z) = T(x, y, z)\, R_z(\gamma)\, R_y(\beta)\, R_x(\alpha) \tag{4.31}$$

$$= \begin{pmatrix} R_{1,1} & R_{1,2} & R_{1,3} & x \\ R_{2,1} & R_{2,2} & R_{2,3} & y \\ R_{3,1} & R_{3,2} & R_{3,3} & z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.32}$$

Similarly, the inverse transformation (from global coordinates to sensor coordinates) is defined using the homogeneous transformation matrix $M'(\alpha, \beta, \gamma, x, y, z)$. The schema of this matrix is given in (4.34), for the calculation of the specific entries, please refer to section B.10.2.

$$M'(\alpha, \beta, \gamma, x, y, z) = R_x(-\alpha)\, R_y(-\beta)\, R_z(-\gamma)\, T(-x, -y, -z) \tag{4.33}$$

$$= \begin{pmatrix} R'_{1,1} & R'_{1,2} & R'_{1,3} & -xR'_{1,1} - yR'_{1,2} - zR'_{1,3} \\ R'_{2,1} & R'_{2,2} & R'_{2,3} & -xR'_{2,1} - yR'_{2,2} - zR'_{2,3} \\ R'_{3,1} & R'_{3,2} & R'_{3,3} & -xR'_{3,1} - yR'_{3,2} - zR'_{3,3} \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.34}$$

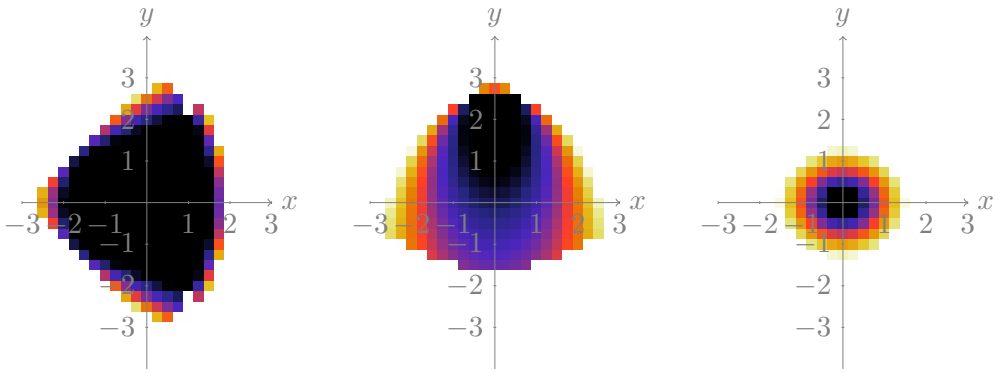
It is worth mentioning that these coordinate transformations can be highly optimized by the use of caching. Fortunately, the sensor mounting positions are usually fixed and thus invariant over time. It is therefore possible to compute the matrices M and M' only once for each sensor and use the pre-computed matrices when coordinate transformations are necessary. Due to the high number of calls to the sensor model in a typical use-case with a particle filter (in every cycle, for every particle, the sensor model needs to be called) this caching can significantly speed up the computation of the sensor model[14].

4.5.3. Visualization of the Confidence Function

The resulting confidence can be evaluated on a grid structure to visualize the sensor model. See figure 4.16 for visualizations of sensor models at different mounting poses using grid structures. In this figure, the color black corresponds to a confidence value

[14] Profiling has shown that especially the calculations of the sine and cosine values in the transformation matrix are very computationally expensive when using their standard C implementations.

of one. The lighter the grid cells are painted, the smaller is the confidence value. White corresponds to a value of zero.



(a) Sensor mounting pose $(-3, 0, 2, 0°, 0°, -60°)^T$, $\alpha_v = 82°$, $\alpha_h = 100°$, $r_s = 5$, $h_t = 1.5$

(b) Sensor mounting pose $(0, 3, 2, -90°, 0°, -45°)^T$, $\alpha_v = 82°$, $\alpha_h = 100°$, $r_s = 5$, $h_t = 1.5$

(c) Sensor mounting pose $(0, 0, 2, 90°, 0°, 0°)^T$, $\alpha_v = 82°$, $\alpha_h = 100°$, $r_s = 5$, $h_t = 1.5$

Figure 4.16.: Confidence map of the 3D-volume sensor model applied to a 2D grid.

4.5.4. From Confidence to Direct Sensor Model

When defining the sensor model for a single PIR sensor, the confidence function is the main ingredient. But first, the possible values a PIR sensor may give need to be defined. Obviously, there are the two events "motion" and "no motion". However, there is a third event, which is called "unknown". This is necessary, as the values of the PIR sensors are sent through a wireless sensor network. Wireless technology is mostly very unreliable technology[15], as messages may get lost and thus the real value of a PIR sensor may be unknown, if it did not send an event for some time. The sensor state is thus defined exactly as in [Arndt 11a] and [Arndt 12c], using a value $z \in \{0, 1, X\}$, representing the values "no motion", "motion" and "unknown", respectively[16].

If a confidence function is used that gives a confidence value between 0 and 1 for each state x of a sensor, then this function can be turned into a direct sensor model that gives the description of the probability $p(z \mid x)$ of sensing the event z in a state x. By the unit measure, it must hold that for a fixed x, the probabilities of all possible events sum up to one:

[15] Unless special care is taken by the wireless protocols to eliminate those issues, e.g. using retransmissions.

[16] This symbol X should not be confused with the definition of the state space X.

$$p(z = 1 \mid x) + p(z = 0 \mid x) + p(z = X \mid x) = 1 \tag{4.35}$$

The confidence function is regarded separately for all the three events. It is first assumed that $p(z = X \mid x)$ (i.e. the probability of having an invalid sensor value in state x) is independent of the state x and is constant over time. This means that a failing sensor is equally likely anywhere in the state space. The probability of a failing sensor can thus be written as follows:

$$p(z = X \mid x) = p(z = X) \tag{4.36}$$

Additionally, if the sensor is providing a valid signal, i.e. if it is either reading $z = 1$ or $z = 0$, then the direct model is defined as follows (with a fixed, but yet unknown normalizer constant η):

$$p(z = 1 \mid x) = \eta \cdot (0.5 + 0.5 \cdot \textbf{Confidence}(x)) \tag{4.37}$$
$$p(z = 0 \mid x) = \eta \cdot (0.5 - 0.5 \cdot \textbf{Confidence}(x)) \tag{4.38}$$

The offset of 0.5 does not matter at this point, but it facilitates the step from the forward sensor model to the marginalized forward sensor model that is introduced later. At this point, it should be noted that if (4.37) and (4.38) are summed together for a fixed state x, they cancel out, so that only the normalizer η remains:

$$\begin{aligned} p(z = 1 \mid x) + p(z = 0 \mid x) &= \eta \,(1 + 0.5 \cdot \textbf{Confidence}(x) \\ &\qquad - 0.5 \cdot \textbf{Confidence}(x)) && (4.39) \\ &= \eta && (4.40) \end{aligned}$$

If additionally, (4.36) is summed to that expression, the result must be 1, according to (4.35):

$$\begin{aligned} \underbrace{p(z = 1 \mid x) + p(z = 0 \mid x) + p(z = X \mid x)}_{=1} &= \eta + p(z = X) && (4.41) \\ 1 &= \eta + p(z = X) && (4.42) \\ \eta &= 1 - p(z = X) && (4.43) \end{aligned}$$

This is an important finding, as (4.43) gives a method to calculate the value of the normalizer constant using the probability $p(z \mid X)$ for a non-valid sensor event as

input. Despite this probability $p(z \mid X)$, it makes sense to further analyze failures of sensors. It is possible that the sensor gives a sensor reading of $z = 0$ even though there actually *is* motion. This is called a *false negative*. On the other hand, it is also possible that a sensor gives a reading of $z = 1$ even if there is *no* motion. This is called a *false positive*. The false negative rate is expressed by the probability that no motion is measured, given the fact that motion is happening. The true positive rate is the inverse of the false negative rate:

$$p_{fn} = p(z = 0 \mid \text{motion}) \tag{4.44}$$
$$p_{tp} = 1 - p_{fn} \tag{4.45}$$

Similarly, the false positive rate is expressed by the probability that motion is measured, given the fact that no motion is happening. The true negative rate is then the inverse of the false positive rate:

$$p_{fp} = p(z = 1 \mid \text{no motion}) \tag{4.46}$$
$$p_{tn} = 1 - p_{fp} \tag{4.47}$$

It is possible to modify the result of the confidence function by multiplying it with p_{tp} (in case of $z = 1$) or with p_{tn} (in case of $z = 0$) to model those two effects. However, care must be taken to not invalidate the relation between $p(z = 1 \mid x)$, $p(z = 0 \mid x)$ and η, as given in (4.40). For this reason, it must hold that $p_{tp} = p_{tn}$, because only then (4.40) stays valid:

$$\begin{aligned} p(z = 1 \mid x) + p(z = 0 \mid x) &= \eta \left(1 + p_{tp} \cdot \mathbf{Confidence}(x)\right. \\ &\qquad \left. - p_{tn} \cdot \mathbf{Confidence}(x)\right) && (4.48) \\ &= \eta \left(1 + (p_{tp} - p_{tn}) \cdot \mathbf{Confidence}(x)\right) && (4.49) \\ &= \eta && (4.50) \end{aligned}$$

4.5.5. Direct Sensor Model for Multiple Sensors

In practice, there is usually more than a single sensor. In such a case, a whole vector $\boldsymbol{z}$ of sensor values needs to be handled. The previous section did only consider the confidence function for a single sensor. If several sensors are used and they are assumed to sense independently of each other[17] then the overall model is very easy to calculate as the product of all single models:

[17] This assumption could in principle be violated because of the wireless communication of sensors on a shared medium, but this is ignored.

$$p(\boldsymbol{z} \mid x) = \prod_{z_i} p(z_i \mid x) \tag{4.51}$$

4.5.6. From Confidence to Marginalized Direct Sensor Model

The (whole) direct sensor model gives $p(z \mid x)$, i.e. a probability density function that describes the distribution of the sensor values z given all possibles states x in which the system might be. In contrast, the *marginalized* sensor model only reasons about a *single* state x_i. The probability $p(z \mid x_i = 1)$ describes the probability for sensing z given the state x_i being "occupied". The state-space is binary, it consists of the elements 0 and 1, meaning "free" and "occupied".

For the situations in which x_i is occupied, i.e. $x_i = 1$, the marginalized sensor model of a single sensor is identical to the non-marginalized sensor model of a single sensor as defined above (using the same normalizer constant $\eta = 1 - p(z = X)$):

$$p(z = X \mid x_i = 1) = p(z = X) \tag{4.52}$$
$$p(z = 1 \mid x_i = 1) = \eta \cdot (0.5 + 0.5 \cdot \mathbf{Confidence}(x)) \tag{4.53}$$
$$p(z = 0 \mid x_i = 1) = \eta \cdot (0.5 - 0.5 \cdot \mathbf{Confidence}(x)) \tag{4.54}$$

For the situations of x_i being free, $x_i = 0$ consequently the following must hold:

$$p(z = X \mid x_i = 0) = p(z = X) \tag{4.55}$$
$$p(z = 1 \mid x_i = 0) = \eta \cdot (0.5 - 0.5 \cdot \mathbf{Confidence}(x)) \tag{4.56}$$
$$p(z = 0 \mid x_i = 0) = \eta \cdot (0.5 + 0.5 \cdot \mathbf{Confidence}(x)) \tag{4.57}$$

The following can easily be verified and is important later on:

$$p(z = 1 \mid x_i = 1) + p(z = 1 \mid x_i = 0) = \eta = 1 - p(z = X) \tag{4.58}$$
$$p(z = 0 \mid x_i = 1) + p(z = 0 \mid x_i = 0) = \eta = 1 - p(z = X) \tag{4.59}$$

4.5.7. From Direct to Marginalized Inverse Sensor Model

Some applications require a marginalized *inverse* sensor model to operate, e.g. the occupancy grid mapping and the occupancy graph mapping. Also, if more than a single sensor should be visualized, the marginalized inverse sensor model can be used to calculate the probabilities for observing a given location x_i in the state space.

The (whole) inverse sensor model gives $p(x \mid z)$, i.e. a probability density function that describes the distribution of all possibles states x in which the system might be, given the sensor value z.

In contrast, the *marginalized* inverse sensor model only reasons about a *single* state x_i. The probability $p(x_i = 1 \mid z)$ (or short $p(x_i \mid z)$) describes the probability for the state x_i being "occupied" given the sensor reading z. The state-space is binary, it consists of the elements 0 and 1, meaning "free" and "occupied". As a result of the binary state, the following relation must hold because of the unit measure:

$$p(x_i = 0) = 1 - p(x_i = 1) \tag{4.60}$$

The marginalized inverse sensor model for a single sensor can be easily created from the direct sensor model for a single sensor (as given by (4.36), (4.37) and (4.38)). The Bayes rule to convert from direct to inverse sensor model needs to be used to calculate the inverse sensor model. The rule in marginalized state notation is given as follows:

$$p(x_i \mid z) = \frac{p(z \mid x_i)\, p(x_i)}{p(z)} \tag{4.61}$$

In order to be able to calculate this term, the probabilities $p(x_i)$ and $p(z)$ need to be known. The value of $p(x_i)$ is called the prior. It defines the knowledge about the single state x_i without having regarded the sensor value. If no information about the state of the environment is available, this prior is usually set to 0.5, indicating that the real state of the state could be either "free" or "occupied".

The value of $p(z)$ is a bit more tricky. Due to the *theorem of total probability*, the following holds:

$$p(z) = \int p(z \mid x_i)\, p(x_i) \mathrm{d}x \tag{4.62}$$

Fortunately, the marginalized state space is binary, i.e. there only exist two values for x_i, so the integral can be easily written as the sum of two terms:

$$p(z) = p(z \mid x_i = 1) \cdot p(x_i = 1) + p(z \mid x_i = 0) \cdot p(x_i = 0) \tag{4.63}$$

By inserting the knowledge of (4.60), it becomes:

$$p(z) = p(z \mid x_i = 1) \cdot p(x_i = 1) + p(z \mid x_i = 0) \cdot (1 - p(x_i = 1)) \tag{4.64}$$

In case $z = X$, the formula simplifies:

$$p(z = X) = p(z = X) \cdot p(x_i = 1) + p(z = X) \cdot (1 - p(x_i = 1)) \tag{4.65}$$
$$= p(z = X) \cdot (p(x_i = 1) - p(x_i = 1) + 1) \tag{4.66}$$
$$= p(z = X) \tag{4.67}$$

If however $z = \star$ with $\star \in \{0, 1\}$, then it reads:

$$p(z = \star) = p(z = \star \,|\, x_i = 1) \cdot p(x_i = 1) + p(z = \star \,|\, x_i = 0) \cdot (1 - p(x_i = 1)) \tag{4.68}$$
$$= p(z = \star \,|\, x_i = 1) \cdot p(x_i = 1) + (1 - p(z = X) - p(z = \star \,|\, x_i = 1)) \cdot (1 - p(x_i = 1)) \tag{4.69}$$

This is due to the property that if summed up they equal $1 - p(z = X)$, see (4.58) and (4.59).

4.5.8. Marginalized Inverse Sensor Model for Multiple Sensors

For multiple sensors, the overall model cannot be calculated as easily as in the case for the direct sensor model, see (4.51). Many of the values $p(x_u \,|\, z)$ and $p(x_v \,|\, z)$ are *not* independent of each other. One sensor usually observes a larger amount of locations of the state space, so that the probabilities cannot be assumed to be independent.

However, the marginalized inverse sensor model for a single sensor can be repeatedly applied for each sensor. The posterior belief (in the formula $p(x_i \,|\, z)$) of one sensor must then be used as the prior belief $p(x_i)$ of the next sensor. The calculation thus becomes a well known recursive Bayesian estimation and can be implemented using a *binary* discrete Bayes filter that reasons over occupied and free states. The algorithm is given in algorithm 4.8.

Algorithm 4.8: Using a binary Bayes filter to calculate the marginalized inverse sensor model for multiple sensors.

Input: Sensor readings $\boldsymbol{z}$, marginalized sensor models $p(z_u \,|\, x_i)$ for all sensors, prior belief about state x_i: belief(x_i)
Output: Posterior belief about state x_i after applying the sensor models.

1: **for all** z_u **do**
2: **if** $z_u = X$ **then**
3: $p(z_u) \leftarrow p(z = X)$
4: **else**
5: $p(z_u) \leftarrow p(z_u \,|\, x_i) \cdot p(x_i) + (1 - p(z = X) - p(z_u \,|\, x_i)) \cdot (1 - p(x_i))$
6: **end if**
7: belief$(x_i) \leftarrow (p(z_u \,|\, x_i) \cdot \text{belief}(x_i)) \,/\, p(z_u)$
8: **end for**
9: **return** belief(x_i)

4.5.9. Conclusion and Other Sensors

To conclude this section on the sensor model, it should be noted that this model is still only a simplification of reality. There are issues and effects that are not handled in this model. For example, the model ignores reflections of infrared radiation, which might happen on walls or furniture. It must also be noted that the sensor model assumes that sensors may look through walls, because obstacles are not handled in any way by this model. Some care has thus to be taken when deciding about the mounting positions of sensors. As long as no "wrong" edge (i.e. one that could not be seen in reality, e.g. due to a wall) of the graph lies in the sensor volume, the model works fine. See also the visualization of the sensor model used in the experiments in this thesis in figure A.10, for an example of "good" sensor placement.

The largest part of this thesis uses the previously introduced wireless sensor nodes for state estimation. Yet, the probabilistic methods are of course not limited to this kind of sensors. The concept is extensible to other sensors that can be modeled in the tracking framework. For example, the next chapter gives an example about how other robots can be used as mobile sensors. The extension of the sensor model in that case is described there (section 5.5.2).

4.6. Single-Target Tracking

For many of the applications that are introduced in the next chapter (when robots come into play), the information where people are currently located is needed. Target tracking is one way to achieve this goal. With the knowledge about Bayesian filters and the appropriate models, that have been introduced in the previous sections, this section presents the application of these methods to the people tracking problem. For the discussions of the first methods and experiments, it is assumed that there is at maximum a single person to be tracked, i.e. there is either none or exactly one person present in the environment. This assumption is relaxed later in section 4.7, which introduces multi-target tracking. However, for now, the presented algorithms represent *single-target* trackers.

4.6.1. Target Tracking using Bayesian Filters

The single-target tracking scenario using Bayesian filters is a straightforward assembly of the previously introduced techniques. In fact, a single Bayesian filter with a motion as well as a sensor model is enough to successfully track a single person in the proposed aware environments. The process is depicted schematically in figure 4.17. Sensor events are fed to a Bayesian filter which accomplishes the tracking using the models and outputs a belief about the position of the person. That belief is then usually used by an application that is not further discussed in this chapter, but only later in chapter 5.

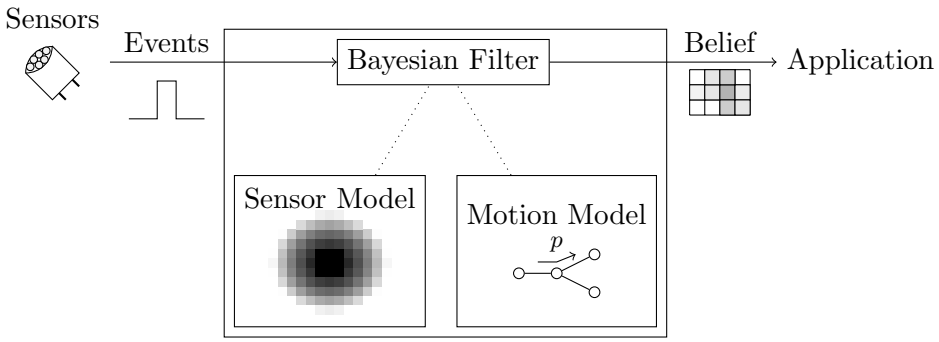


Figure 4.17.: Schematic representation of the single-target tracking method.

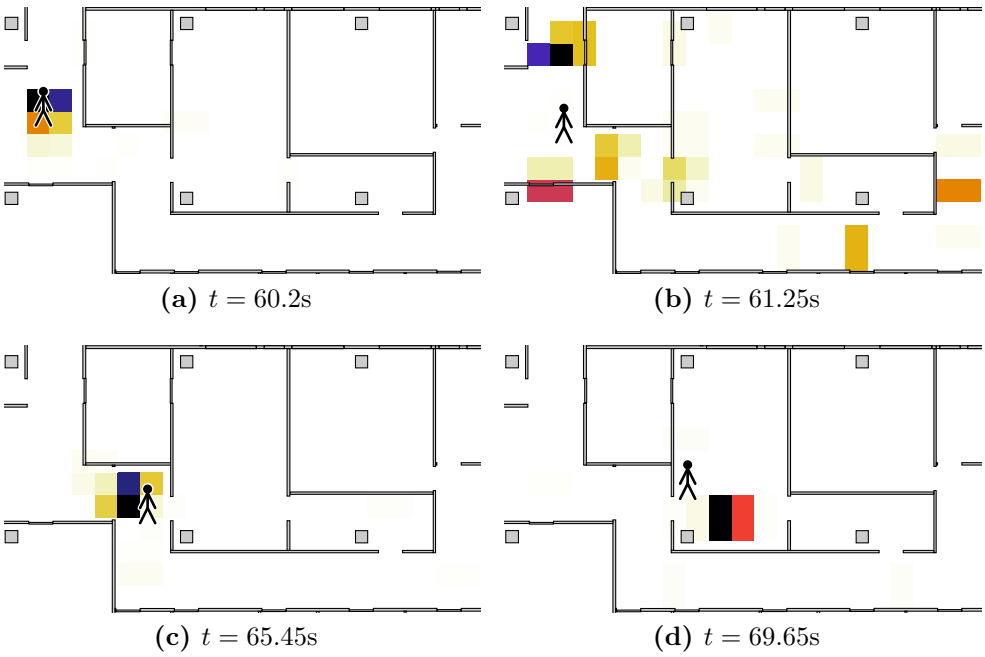
In the scope of this thesis and the prior work leading to it, people tracking using a particle filter (as introduced earlier in section 4.2.4) has been established. The tracking of single targets using a particle filter and the previously introduced models (the sensor model for passive infrared sensors on wireless sensor nodes, see section 4.5 and the basic Brownian motion model, see section 4.4) with a graph-based state space (see section 4.3) in an example environment with AmICA sensor nodes has been tested and evaluated in several experiments.

4.6.2. Experiments

In the following, one of the conducted experiments is described and evaluated in detail. The experiment was set in the office environment of the Robotics Research Lab at the University of Kaiserslautern. A total of ten AmICA wireless sensor nodes (see section A.1 for a detailed description of these nodes) had been installed in the environment during the experiment. The setup of this and most of the upcoming experiments (unless differences are explicitly noted) is identical. For this reason, it is *not* replicated in detail here. Instead, the reader is directed to the appendix, section A.3, for an exact plan of the environment as well as for the exact sensor placements and parameters chosen in the models.

For the single-target tracking scenario that is discussed in this section, data has been gathered by the author while walking through the environment with a hand-held camera. The ground truth has been restructured by manually analyzing the videos and labeling key points of the trajectory. Positions between two manually labeled points are linearly interpolated. Tracking has then later been performed off-line using the recorded data from the sensor nodes[18]. In the experiment that is shown in this section, the number of particles was fixed to 250.

To give the reader a visual clue what the tracker is doing, a visualization of a tracking



(a) $t = 60.2$s **(b)** $t = 61.25$s

(c) $t = 65.45$s **(d)** $t = 69.65$s

Figure 4.18.: Visualization of some snapshots of a tracking sequence by a single target tracker. The belief is visualized by mapping it to a grid structure. The darker the colors of the cells, the higher the probability of the person being present.

sequence is shown in figure 4.18. This sequence only shows few snapshots in time, as the complete sequence, created with a frequency of about 20Hz, contains way too many situations to be replicated here. A more detailed view on that sequence which shows more situations is given in section C.1. In all these and the following figures that are used to visualize tracking sequences, a mapping of the belief to a grid representation has been used, as this offers a very intuitive way to show how the belief is distributed. The exact process is described in detail in section 4.3.2. For the visualization, the "full" mapping of weights to grid cells with an additional scaling of the values into the interval $[0, 1]$ has been applied. This ensures that always the whole dynamic range offered by the visualization is used and different probabilities can be easily distinguished. However, this also implicates that, if the reader should be able to read the exact value of a grid cell, a different legend that maps from colors to values would have to be given for each shown situation. As this is not beneficial to the presentation of the results, this has not been done. The visualizations should instead be seen as a help for *qualitative* analysis of the results[19]. Everything the reader needs to know is: The darker the color, the higher the probability of a person being present

[18] Although this experiment did the tracking off-line, on-line tracking with the same parameters is perfectly feasible. On-line tracking is used later in the experiments with the robot.

[19] A *quantitative* analysis has also been performed and the results are discussed very soon.

at that cell.

Coming back to figure 4.18, in the first panel, the person (its ground truth is indicated by the pictogram) has just triggered a sensor. The belief of the tracker is concentrated at that location. Shortly thereafter, as indicated by the second panel, the person leaves the range of that sensor. Naturally (as dictated by the motion model), the belief spreads out through the environment in the three possible directions the person could take. The belief is focused once more in the third panel, when the person triggers another sensor mounted there. The fourth panel shows a similar situation as the third one, the person triggers a sensors and the belief is concentrated around the person.

To evaluate the tracking performance in terms of numbers, the tracking error δ and the sample standard deviation s of the tracking results has been calculated similar as previously done by the author in [Arndt 11a] and [Arndt 12c]. To calculate these values, the sample mean $\overline{\boldsymbol{x}}$ of the belief needs to be defined and calculated first. Although the tracker operates on a graph-based state space, to calculate the mean, the particles' coordinates are first converted into Cartesian coordinates and the mean is calculated in this space. See (4.70) for the calculation of the sample mean over all particles, indexed with i using the Cartesian particle position $\boldsymbol{x}_i$ and the particle weight w_i:

$$\overline{\boldsymbol{x}} = \sum_i w_i \cdot \boldsymbol{x}_i \tag{4.70}$$

This assumes that the weights w_i of all particles sum up to 1. If this is not satisfied, the result must be divided by the sum. Using the knowledge of $\overline{\boldsymbol{x}}$, the tracking error can be calculated as the Euclidean metric of the difference between the ground truth and the sample mean:

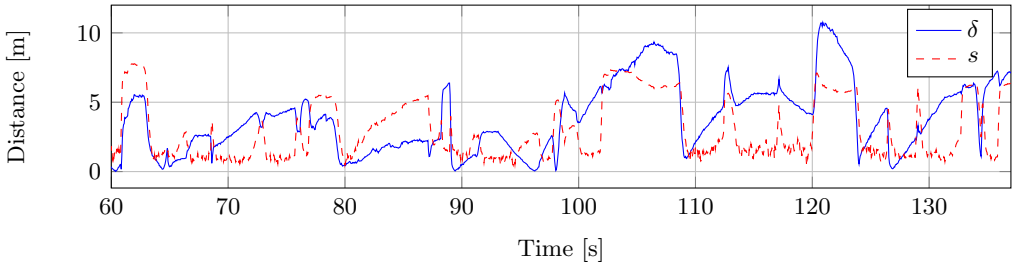
$$\delta = \|\boldsymbol{x}_{\text{true}} - \overline{\boldsymbol{x}}\| \tag{4.71}$$

The (uncorrected) sample standard deviation s is computed according to:

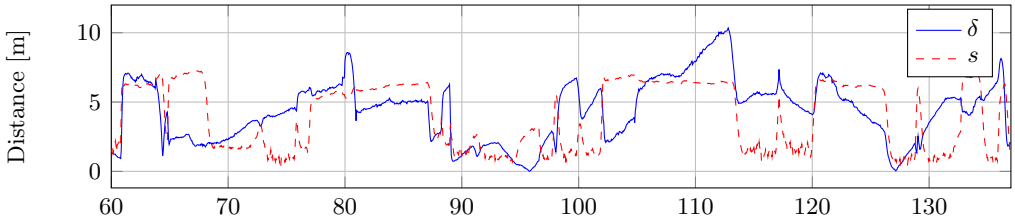
$$s = \sqrt{\sum_i w_i (\boldsymbol{x}_i - \overline{\boldsymbol{x}})^2} \tag{4.72}$$

For an excerpt of the full experiment, these values are plotted in figure 4.19. The timestamps in these plots correspond to the ones given in figure 4.18.

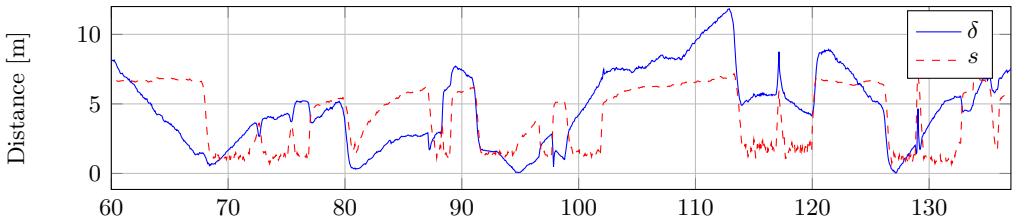
Interestingly, one node (with identifier 1) did fail for the whole experiment. This was not planned at the beginning of the experiment, but thanks to the failure-resistant



(a) No further disabled nodes.



(b) Nodes with identifiers 7 and 255 disabled.



(c) Nodes with identifiers 7, 9, 10 and 255 disabled.

Figure 4.19.: Analysis of tracking error δ and sample standard deviation s over time while tracking a single person. The plots show only excerpts of a larger experiment.

sensor model, this did not cause the experiment to fail. It only degraded the expected performance, as less sensor information than planned was available.

To analyze this *graceful degradation* further, the tracking has not only been performed with all the recorded information available, but also with reduced data, in which several sensor values have been artificially "hidden" from the tracker, thus reducing the effective number of active nodes. See figure 4.19b for an example in which two nodes have been artificially disabled. To pursue this idea even further, in figure 4.19c, two more nodes have been disabled, leaving only five working nodes active. By comparing these three figures, it can be concluded that a more dense sensor instrumentation does increase the accuracy of the target tracking.

The additional visualizations in section C.1 can be used as a supplement to the plots presented here. In the appendix, the results of the three tracking scenarios (with varying numbers of active sensors) are given next to each other, enabling an easy comparison of the situations.

4.7. Multi-Target Tracking

In the previous section, it has been assumed that exactly one person was occupying the environment. While this might be true in artificial scenarios, it is of course not true in general. In this section, this assumption is relaxed and multi-target tracking is introduced.

In principle, the task of a multi-target tracker is similar to that of a single-target tracker. In case of multi-target tracking, however, the output of the tracker is different. It is no longer a *single* belief about the state of a *single* target. Instead, if the tracker is able to distinguish between different targets, it outputs *one* belief for *each* target. Additionally, or if the tracker does not maintain single beliefs, a joint belief over all target states could be available as an output, as indicated in figure 4.20.

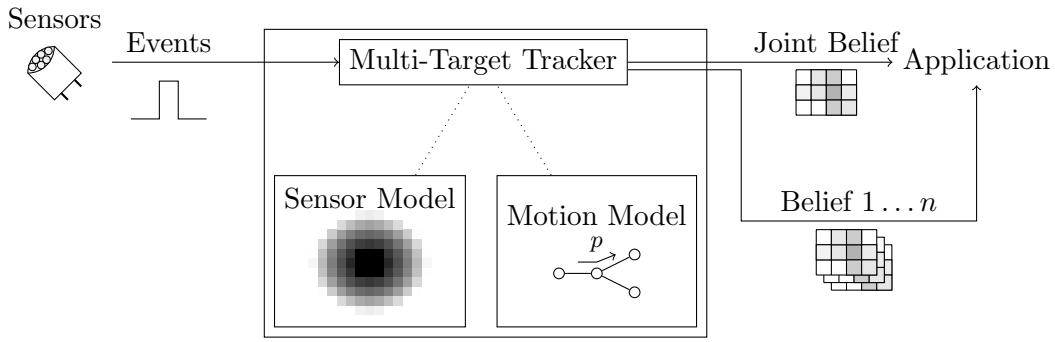


Figure 4.20.: Schematic representation of a generic multi-target tracker which outputs individual beliefs for each target and a joint belief over all targets.

Unfortunately, the evolution from single-target to multi-target tracking is not as trivial as one might imagine at first. It is not possible to just create multiple instances of a single-target tracker to obtain a multi-target tracker. Instead, a new problem called the *data association problem* arises.

4.7.1. Data Association

The data association problem is a fundamental problem of many (but interestingly not all[20]) approaches to multi-target tracking. The data association problem can be explained using the abstract example in figure 4.21. In the example, there is a new measurement m and there are two tracks, t_1 and t_2, which represent two targets. The distance of that measurement to both tracks is very similar – so which track should it be associated with?

But these are not the only possibilities – it could also be the case that this measurement belongs to a completely new track t_3 or that it constitutes a false measurement (i.e.

[20]The probability hypothesis density filter introduced later, does not need data associations.

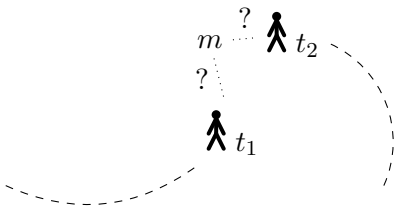


Figure 4.21.: Visualization of the data association problem for an abstract scenario with two tracks (targets) and one observation.

one that must not be associated with any track at all). Depending on the application of the tracker, a wrong association might lead to a confusion of tracks, the loss of a target or the introduction of spurious tracks that do not represent a real target. This small example illustrates well that the data association step in multi-target trackers is a critical component which has led to several methods of dealing with the problem.

A very basic technique to limit the amount of possible associations (which has a high complexity in the number of existing tracks and observations) is *gating*. Gating uses a metric to provide a list of candidate observations that a track can be associated with. In figure 4.22, this process is illustrated in the two-dimensional Cartesian case with the Euclidean norm as metric.

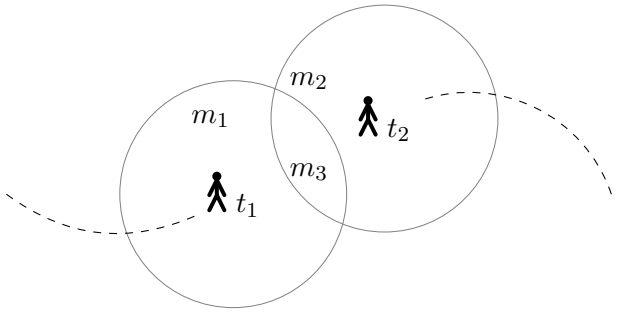


Figure 4.22.: Visualization of the gating procedure with the Euclidean norm as metric.

Both targets t_1 and t_2 have a circular gate (given by the Euclidean norm) around their estimated positions. According to these gates, measurements m_1 and m_3 are candidates to be associated with t_1 while m_2 and m_3 are candidates for t_2. It is worth noting that this does not mean that the tracks are later really associated with one of these measurements, it may e.g. be possible that *all* measurements may be classified as false measurements or that all initiate new tracks. This decision can only be made in the real data association step, not in the gating step.

4.7.2. Related Work and Anonymous, Sparse Sensor Instrumentation

In section 3.3 of the thesis, it has been noted that the proposed system shall work with a *sparse* instrumentation of the environment with *anonymous* sensors. The previous section has already shown that *single* target tracking in such environments is in principle possible, although the accuracy of the tracking results depends on the actual density of instrumentation.

When it comes to multi-target tracking, a large amount of work that describes applications of multi-target trackers can be found, but much of that work makes actually use of rich and/or identifying sensor data as input to the tracking process. This is obviously contrary to the goals of this thesis.

The work that probably comes closest to this work and which forms important foundations, is the work by Schulz et al. [Schulz 03]. Their work uses a combination of anonymous and non-anonymous (which they call *ID-Sensors*) that feed data to Rao-Blackwellized particle filters to estimate the positions of multiple people in a smart environment. The identification sensors are infrared and ultrasound badges that need to be *actively* worn by the people in the environment, the anonymous sensors are two wall-mounted laser rangefinders. It is unclear whether these rangefinders are able to observe the whole office environment, but it can be assumed that a large area is covered (see [Schulz 03, Fig. 3]), so the sensor instrumentation can be assumed to be neither completely anonymous nor really sparse.

The work by Särkkä et al. [Särkkä 07] proposes the *Rao-Blackwellized Monte Carlo data association (RBMCDA)* algorithm to track an "unknown and time-varying number of targets". While that work includes simulation experiments which make use of an *anonymous* sensor (which can only detect the angle of a target) that sensor is able to observe the *whole* state space, meaning it does not fit the definition of a sparse sensor in this work. Additionally, the hypothetical assumption to use sensors of such type to fully observe a smart environment, would be infeasible in terms of installation effort and costs.

Kjellström (formerly Sidenbladh) [Sidenbladh 03] presents a particle filter implementation of a *probability hypothesis density (PHD)* tracker (see below for further details) to also track multiple targets. In her work, the performance of the implementation has been tested in a (simulated) vehicle tracking scenario which uses "a noisy version of the real state" [Sidenbladh 03, 5.1] as input-vector to the tracking problem. The observation contains vehicle position, speed and direction, thus this work does not qualify as using sparse sensors. Also the anonymity of a sensor measurement is limited, as it contains rather detailed information about the target state. Nevertheless, the PHD tracking approach using particles is quite appealing and is evaluated later for the setting of this thesis. Regarding PHD tracking with particle filters, Sidenbladh is not alone in the field and about at the same time, Vo et al. [Vo 03] have also come up with a sequential Monte Carlo implementation for which they also provide simulation results, however they also use the (noisy) position of the targets as input, which can

be observed throughout the state space, thus the work is – as seen from this angle – very similar to the one of Kjellström.

An extreme example of a work that is related, but constitutes a strong contrast to this work is the one by Han et al. [Han 12]. Their work describes tracking of humans using RGB-D sensors, i.e. cameras with additional depth information. While the authors allow an instrumentation of the environment, which is not completely rich in terms of observability [Han 12, I.], these types of sensors are obviously non-anonymous. However, the authors do not even claim that they aim to have privacy in their smart environments, but their goal is to explicitly identify users [Han 12, p. 255], so this work builds upon a very different philosophical standpoint and has thus different goals.

Starting from the findings in the literature, two different multi-target trackers have been chosen, implemented, adapted and tested with the application scenario of this thesis: The multiple hypothesis tracker (MHT) and the probability density hypothesis (PHD) tracker.

4.7.3. Multiple Hypothesis Tracker (MHT)

The multiple hypothesis tracker is a multi target tracker that is able to maintain individual beliefs for multiple targets. It dates back to at least 1979, when Reid published his implementation [Reid 79]. The basic idea of this method is to defer the data association step until there is enough information available to achieve a good association between sensor events and tracks. The algorithm can attain this goal by not only preserving a *single* hypothesis of association, but by preserving and updating a set of multiple hypothesis (hence the name) over a period of time. Depending on the application and the choice of the parameters, the computational requirements for a Multiple Hypothesis Tracker can grow large very quickly, as the tree of hypothesis expands fast if unlikely hypothesis have to be kept for a longer time.

The MHT works in several steps: First, *gating* is applied to the current measurements to limit the number of possible associations from measurements to tracks. Only the most likely (i.e. close) measurements can be associated to existing tracks. For each existing track, a new hypothesis is then formed, by assigning the (gated) measurement to that track. Additionally, new hypothetical tracks are created that are assigned to the measurement. As the MHT also allows false measurements, each existing track is also assigned to the *dummy* measurement which means that that target was not observed at all at the current time step. All of the hypothesis are then updated according to the models.

For this work, a track-oriented Multiple Hypothesis Tracker, as described by Blackman [Blackman 04], has been implemented. Each hypothesis consists of an instance of a single-target tracking algorithm (in this case a particle filter with $N = 100$ particles). Gating is implemented by checking if the score of a particle set according to a measurement is greater as some threshold. The score is computed by applying the

motion model for the current measurement for each particle in the particle set. The result is a normalized weight which is checked against a threshold in the experiments. The implementation has a depth of 4 scans, i.e. the hypothesis will keep a history of the last four measurements.

In artificial examples with manually crafted sensor events, this tracker did perform not entirely bad, but in real scenarios with quickly changing sensor values, the implementation started to fail very quickly. The issue is most likely due to the small scan depth[21] and the sparse sensor instrumentation (which leads to a high probability of *not* observing a target in a scan). For this reason, the multiple hypothesis tracker has *not* been further considered to be suitable for multi-people tracking in the proposed environments.

4.7.4. Probability Hypothesis Density (PHD) Filter

While some multi-target tracking approaches basically take single target tracking filters and use several instances of these (with some external data association logic) to be able to track multiple targets, the probability hypothesis density (PHD) filter works differently. Instead of maintaining the exact belief for each of the targets (and thus also the associations between target and belief), the PHD filter only tracks the *first moment* (the mean or the expected value) of the joint probability distribution of *all* targets [Sidenbladh 03].

Similar as with the classic recursive Bayesian filters, a closed-form of the exact PHD filtering problem is often unavailable [Vo 03]. For this reason, just as in the classic particle filter, the PHD filter is commonly implemented using sequential Monte Carlo methods [Sidenbladh 03, Vo 03]. For this thesis, the method proposed by [Sidenbladh 03] has been implemented. It uses a variety of models and parameters which describe the system (the environment and the behavior of people). The motion model and the sensor model of the PHD filter are identical to those used in the other probabilistic filters. What is new for the PHD filter is a model which describes the birth PDF according to (4.73):

$$f_{x_t \mid z_{t-1}}(x_t \mid z_{t-1}) \tag{4.73}$$

This PDF defines where new targets are likely to appear, given the previous observations z_{t-1}. For simplifications, this distribution may be set to a uniform distribution (as done in this work). Despite the birth PDF, there are some more additional parameters that need to be carefully adjusted for the multi-target tracking using the PHD filter. These parameters are the probability of target birth p_B, the probability of target death p_D and the probability of false negatives p_{fn}. The last parameter is very important in the sparse sensor application scenario, as there will be *many* false

[21] An increase in scan depth will increase the computational complexity by allowing a much larger number of association hypothesis. It might therefore no longer be feasible to execute the tracker on-line.

negatives (i.e. non-observed targets), because the sensor coverage is not complete. In fact, the parameter p_{fn} could be calculated by the ratio of non-observed to observed space.

4.7.5. Experiments

For the experiments, the following parameters have been used: The number of particles has been set to $N = 250$, just as for the single target tracking example[22]. To account for a large amount of unobserved space, the probability of false negative observations has been set to $p_{fn} = 0.5$. The probability of a target death has been set to $p_D = 0.01$ and the probability of target birth is computed just as in [Sidenbladh 03, 5.4] as:

$$p_B = p_D^{1-p_{fn}} \tag{4.74}$$

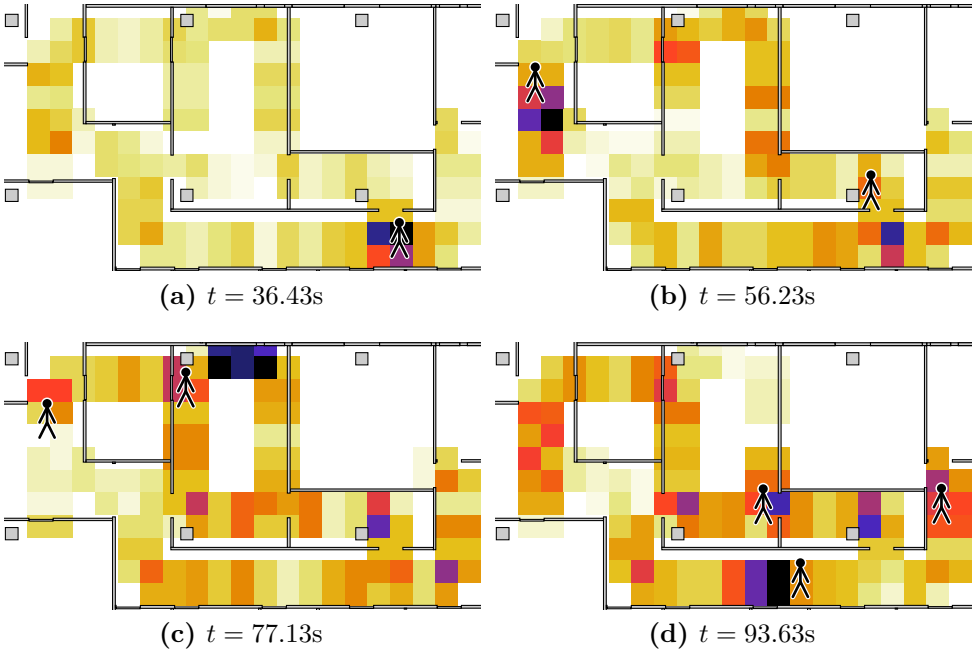
Similarly as done for the single-target tracking case, in the same setup, also sensor data and ground truths from up to three people in the environment have been recorded. The data has been gathered by the author and two volunteers who walked through the environment with a total of three hand-held cameras. The ground-truth data has been manually labeled from the video images, just like in the single-target tracking case.

The data has then been fed off-line[23] into the PHD filter (as described above). In contrast to the single-target tracking example, the accuracy of the tracking cannot be that easily evaluated. This is also due to the properties of the PHD filter which does only maintain a joint belief and no individual beliefs for all targets. For this reason, the evaluation of the results is only done qualitatively using snapshots of an example sequence. This sequence is depicted in figure 4.23.

There is one aspect that can be noted from these visualizations, compared to the visualization of the single target tracking in figure 4.18, right away: The output beliefs of the PHD filter are much less concentrated than the ones of the single-target tracker. This circumstance is clearly visible, even if there is only a single target being present, e.g. in the first panel. The reason for this is twofold. First, the birth cloud creates new particles, uniformly distributed in the environments to be able to cope with targets appearing anywhere in the state space. But also the rather high probability for false negatives p_{fn} plays an important role for this effect, as it states that even if there is no "event" being observed right now, the target may still exist somewhere in the state space. With a sparse sensor instrumentation like in the proposed scenario, the

[22] However, note that this number is not the absolute number of particles used for estimation, but a parameter that is used for the birth cloud and is better described as "number of particles per target".

[23] The same remarks as for the single-target tracking applies: It is feasible to execute the tracker on-line. This is done in the later experiments that have been conducted for application examples.



(a) $t = 36.43$s **(b)** $t = 56.23$s

(c) $t = 77.13$s **(d)** $t = 93.63$s

Figure 4.23.: Visualization of some snapshots of a tracking sequence by a probability hypothesis density tracker. Darker grid cells represent a higher probability of a person being present.

probability is indeed very large that a person walks around for some time without being detected by a sensor node.

The PHD filter sacrifices clearly distinguishable beliefs, as compared to the single-target tracking scenario. On the other hand, it is indeed able to track multiple people in the environment and can also cope with a varying number of targets. The reader can also verify this using the additional tracking sequences that are given in section C.2. These visualizations also show that the belief of the PHD filter is still rather "determined", as compared to the pessimistic occupancy graph mapping, that is introduced in the next section.

4.8. Occupancy Graph Mapping

The outputs of the previously introduced single and multi-target trackers (i.e. the beliefs that describe the probabilistic position of the people) were the main ingredient to all the methods that have improved robot behavior in the sense of this thesis[24] in the previous publications of the author. Often, these beliefs are *optimistic*, meaning they mark space as non-occupied, although it is not completely clear that it really *is*

[24]The algorithms are introduced later in chapter 5.

free. While this constitutes no grave issue for some applications (e.g. the risk-reduced path planning in section 5.2 or the predictive path planning using the tracking data in section 5.3), for other applications (e.g. the virtual obstacle avoidance sensors, see section 5.1) it is favorable to *not* rely on this optimistic tracking data, but to use *pessimistic* estimations of the state instead. The reason is to ensure safety, even in the case of a failing tracker. This pessimistic estimation is realized using the occupancy graph mapping that is described in this section.

4.8.1. Relation to Target Tracking

Similar to the concept of occupancy grid mapping (see section 2.1.1.1), the graph that has been used for tracking in the previous sections can also be used as an *occupancy graph* which encodes where the environment (as represented by the graph) might be occupied.

This process is related to the single and the multi-target tracking that has been discussed so far, because it uses some of the components that are also needed for the tracking. There are however also a number of important differences. The obvious one is the fact that it cannot be used for *tracking*, because, just as with the occupancy grid maps, it does not estimate the multi-dimensional "map" directly, but only the probability of a certain "cell" to be occupied. For this reason, the dependencies between single "cells" (or as can be seen later – in this case particles) are not considered, and a motion model cannot not applied.

How is the occupancy graph mapping relevant if there are already methods for tracking available? First of all, there are situations in which the exact (or, in probabilistic language, the most probable) positions of people in the environment are not important, but merely the fact that some position is either occupied or free is interesting (this is exactly the knowledge that the occupancy graph mapping provides). One example is the use of the information as a virtual obstacle avoidance sensor, as discussed later in section 5.1. In this application, the exact distribution of the belief does not matter, the only important information is whether there exists a person close to the path of the robot.

Additionally, the tracking is always only as good as the models that are given to it (especially the motion model is of great importance). There are cases where the tracking just fails to exactly track a person and especially in multi-target tracking, the estimation of the correct number of targets is challenging. In these situations, it is better to have a "pessimistic" view on the current state of the environment than one that may be inaccurately classifying space as free. This is highly relevant if the uses of the (tracking) data are safety-critical, such as in the abovementioned virtual obstacle avoidance application. If the safety of a system depends on the data gathered through the sensors distributed in the environment, it is better to use the pessimistic solution that marks more area as "occupied" than to rely on the tracking of people which may be inaccurate.

4.8.2. Occupancy Graph

Simply speaking, an occupancy graph consists of two components – structure and data. The structure is defined by an undirected graph, while the data is represented using particles. This makes the occupancy graph identical to the graph-based state space introduced earlier (in section 4.3). An important difference though, is the *placement* of the particles. While the particles in the tracking applications are *moved* along the edges of the graph (by the motion model), the particles of the occupancy graph are *fixed*. Their positions are (ideally) initialized equidistantly once at the beginning and are not changed during runtime of the occupancy graph mapping algorithm.

Formally, the occupancy graph consists of an undirected graph $G = (V, E)$ defined like in section 4.3 and a representation of the belief in form of a set of fixed, uniformly distributed, weighted particles[25] $P \subset (E \times \mathbb{R} \times \mathbb{R})$. A particle $p = (e_{ij}, t, w)$ contains information about the state (the edge e_{ij} and the parameter t, see section 4.3) and the weight w. For the process of occupancy graph mapping, a marginalized inverse sensor model is needed that is used to adapt the weights of the particles. This can e.g. be the one that has been introduced earlier for the PIR sensors in section 4.5. An update rule is used to adapt the particles' weights if there are sensor events. To allow for dynamically changing situations, the minimum and maximum probabilities of each particle should be limited so that even a particle which has been "free" for a long time can be updated to "occupied" in reasonable time.

An illustration of the occupancy graph itself as well as the mapping process is given in figure 4.24. The uniformly distributed particles are visualized on the edges of the

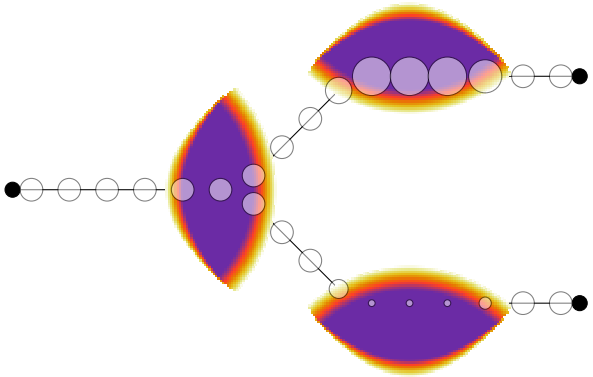


Figure 4.24.: Occupancy graph mapping with uniformly distributed particles, visualization of the sensor model and the update using three different sensor states.

graph with circles. Their diameter is proportional to the particle weight. The figure also visualizes the sensor model of the three sensors with the area they cover. The update process is visualized by feeding artificial sensor data to the sensors for the

[25] Possibly, there are other methods to represent the belief on a graph-based state space, but these are not be considered in this thesis.

visualization. The left sensor does not provide a valid signal and does thus *not* change the particle weights. The upper sensor indicates "occupied" and the lower sensor indicates "free", both of them thus do modify the particles' weights (their sizes in the figure).

4.8.3. Particle Initialization

As already mentioned, the occupancy graph mapping uses (fixed) particles on the already-known graph to represent the current state. At the beginning, these particles need to be initialized. The initialization procedure distributes N particles approximately uniformly (i.e. equidistantly) on the graph and initializes each weight with $w = p(\text{occupied}) = 0.5$. To distribute the particles approximately equidistantly on the graph, the following procedure is used:

1. Compute the sum of all edge lengths of the graph $G = (V, E)$:

$$\Sigma_E = \sum_{e_{ij} \in E} l_{e_{ij}} \tag{4.75}$$

2. Compute the particle density:

$$\rho = \frac{N}{\Sigma_E} \tag{4.76}$$

3. For each edge $e_{ij} \in E$ assign $n = \lfloor \rho \cdot l_{e_{ij}} \rfloor$ particles to that edge. The parameter for linear interpolation t should be chosen so that the particles are distributed equidistantly, locally to that edge. The value of t can be computed for the i-th particle ($i \in [1, n]$) on the edge using: $t = \frac{i-0.5}{n}$

It must be noted that this procedure can introduce inaccuracies due to rounding and the equidistant property may be violated. There are situations where it is impossible to fulfill this property, e.g. when more than two edges are connected through a vertex. If however the number N is large, the inaccuracies quickly become negligible. An example of an initialization can be found in figure 4.25. Here the number of particles has been set to $N = 10$, the edges have lengths of $l_{e_{12}} = \sqrt{2}$ and $l_{e_{23}} = 2$, resulting in a particle density of $\rho = 10/(2 + \sqrt{2})$. This yields $\lfloor (10 \cdot \sqrt{2})/(2 + \sqrt{2}) \rfloor = 4$ particles on the first edge and $\lfloor (10 \cdot 2)/(2 + \sqrt{2}) \rfloor = 5$ particles on the second edge. It can be seen that this is an example where the inaccuracies due to rounding are quite dominant as only 9 instead of 10 requested particles are assigned.

4.8.4. Update of Particle Weights

During operation, the location of the particles stays fixed, however their weights are adapted according to the incoming sensor information. The update process makes use

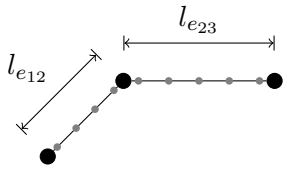


Figure 4.25.: Initialization of the particles for the occupancy graph mapping.

of the marginalized inverse sensor model developed earlier (see section 4.5.7), but it does not need a motion model, as it only reacts on sensor measurements. The process of updating the weights is very similar to that of occupancy grids (see [Thrun 05, 9.2]) and to the method that creates the marginalized inverse sensor model for multiple sensors in section 4.5.8: The core of the update process is a binary Bayes filter.

When the occupancy graph mapping algorithm is supplied with new sensor data, the binary Bayes filter as given in algorithm 4.8 on page 66 is applied to each particle in the particle set to update its weight. The prior which is supplied to the algorithm is in this case the particle's prior weight, before updating. In order to never let the estimation be "too sure" about the state of the environment, the lower and upper probabilities of a particle are capped to $p_{\min}$ and $p_{\max}$, respectively. In the following examples and experiments, these values have been set to $p_{\min} = 0.1$ and $p_{\max} = 0.9$. Without this limitation, it might take very long to change the estimated state from "occupied" or "free" to the other. This is in contrast to the typical occupancy *grid* mapping algorithms, which aim to estimate the state of a *fixed* environment, whereas in the occupancy *graph* mapping, the environment is *dynamic*.

4.8.5. Example Update Sequence

An exemplary sequence of updates applied to an occupancy graph can be seen in figure 4.26. This example uses the same graph structure and the same sensor model as previously shown in figure 4.24. Before the first update, at $t = 0$, the situation shows a non-uniform distribution of the belief. There is one area with higher probability (at the top) and one area with lower probability (at the bottom). The occupancy graph is then successively updated several times with new sensor information. The sensor information states "motion" for the sensor at the left and at the bottom while the sensor at the top states "free". It can be seen that the probability changes over time as more updates are made.

4.8.6. Experiments

The same data used for the previous evaluation of the probability hypothesis density filter (see section 4.7.4) has been processed using the occupancy graph mapping process. Initially the particles have all been initialized to have a probability of $p = 0.5$.

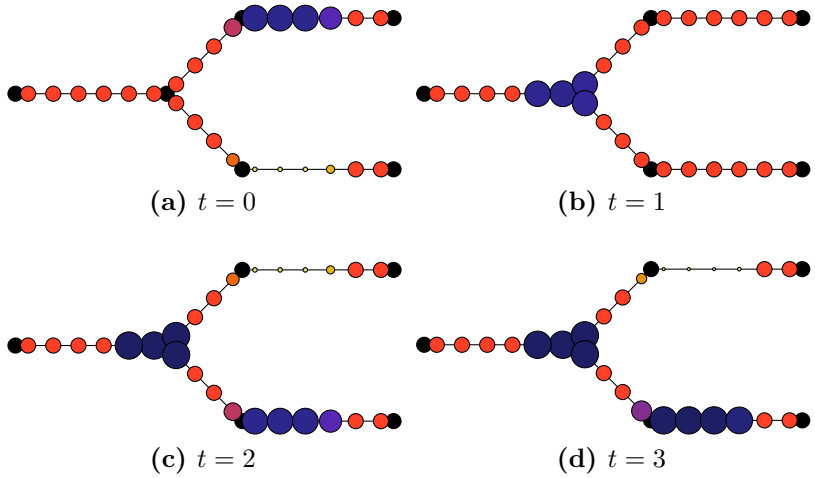
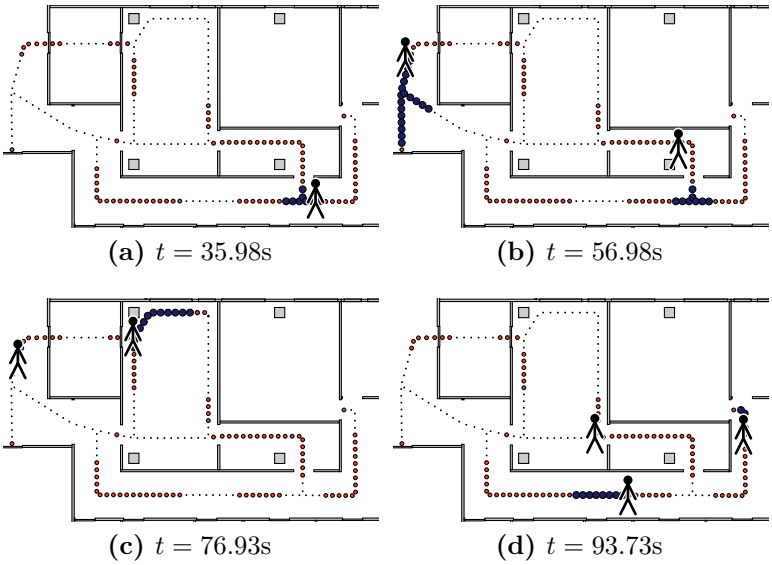


Figure 4.26.: Example update sequence for an occupancy graph.

The result for similar points in time as for the PHD filter (cf. figure 4.23) is given in figure 4.27. In this figure, the estimated state is visualized using particles that are drawn in different size and color to show their weight (i.e. the associated probability for representing an "occupied" state). The same situations are also depicted in figure 4.28 where the belief has been mapped to a grid structure, so that the result is easier to compare with that of the trackers.

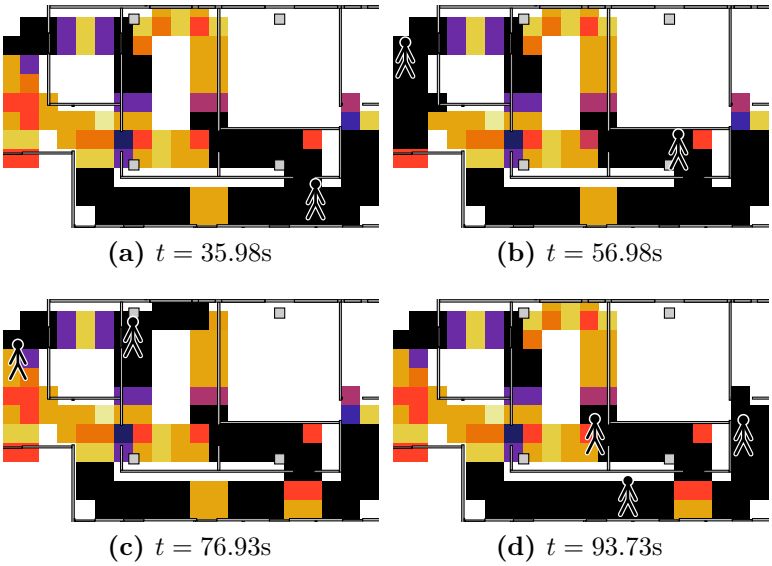
From the figures, it can be seen that the occupancy graph mapping does not perform *tracking* in the sense that missing sensor observations are "filled" using a sensor model. The "pessimistic" property of the method can also be seen, as only those areas that are clearly marked "free" by the sensors are represented with lightly-weighted particles. For more detailed visualizations and side by side comparisons of the results of the occupancy graph mapping with the output of a PHD filter, please refer to section C.2 in the appendix.



(a) $t = 35.98$s **(b)** $t = 56.98$s

(c) $t = 76.93$s **(d)** $t = 93.73$s

Figure 4.27.: Example update sequence for an occupancy graph in reality. The estimated state is directly represented using particles of different weight.



(a) $t = 35.98$s **(b)** $t = 56.98$s

(c) $t = 76.93$s **(d)** $t = 93.73$s

Figure 4.28.: Example update sequence for an occupancy graph in reality. The estimated state is represented by first mapping the particles to a grid representation. Darker grid cells represent a higher probability of being occupied. Particle weights that represent a probability ≥ 0.5 mark a grid cell as "occupied".

5. Safe and Cost-Efficient Navigation

The previous chapter has introduced the methods and formal foundations that are needed for the rest of this thesis. This includes the modeling of the state space using a graph and the processing of external sensor data with its help, especially using the concept of occupancy graphs and in the form of single and multi-person tracking. Almost everything contained in that chapter is independent of robotics.

This chapter now steps more deeply into the subject of the thesis, which also involves mobile robotics. By making use of the methodologies acquired in chapter 4, this chapter discusses several aspects that lead to reaching the goals of the thesis, as defined in detail in section 3.3.

5.1. Virtual Obstacle-Avoidance Sensors

One of the first application ideas to make use of external sensor information was the one of creating a virtual distance sensor for a mobile robot. The essence of this application was contained in the author's Master's thesis [Arndt 11a] and has been published in the proceedings of the 7th German Conference on Robotics [Arndt 12c]. Ultimately, this idea has led to the creation of this thesis.

The basic concept behind it is to "mount" a forward-facing virtual distance sensor on the robot, "looking" at the output of the state estimation process. The goal of this sensor is to be able to increase the "sight" of the robot to places further away, so that the robot is able drive at higher speeds, while still being able to stop in time if dynamic obstacles (i.e. people) come close. A typical robot usually has a number of local sensor systems attached, to allow the robot to localize and to avoid obstacles, see section 2.1. Depending on the tasks of the robot and the budget available, their number, precision as well as their range may vary. For this first and also the following applications, the mobile robot Artos has been used, unless noted otherwise. This robot is a small multi-purpose indoor robot. Its specifications and a further description can be found in the appendix, see section A.2. The local sensors of the robot are all rather short ranged and additionally, by their principle of measurement, they are unable to perceive obstacles behind walls or around corners. If a robot should be safe, in the way that it can always avoid collisions with obstacles, it must be able to come to a complete stop when a collision is imminent. The time to come to a stop is dependent on the speed of the robot[1]. Driving faster requires a longer sensing range[2],

[1] And its braking deceleration.
[2] Or a higher braking deceleration.

however problematic situations like corners or curves that allow no direct line of sight may *not* be resolved by having sensors with longer range.

It is therefore an interesting idea to use the results of the state estimation, as introduced in the previous chapter, to create a *virtual* sensor in the robot's control system that reacts appropriately on these dynamic, human obstacles. See figure 5.1 for an illustration of the idea. In the shown situation, the local sensors of the robot are impaired by the corner next to which the robot is currently situated. In contrast to that, the field of "view" of the virtual obstacle avoidance sensor is able to "look" around the corner.

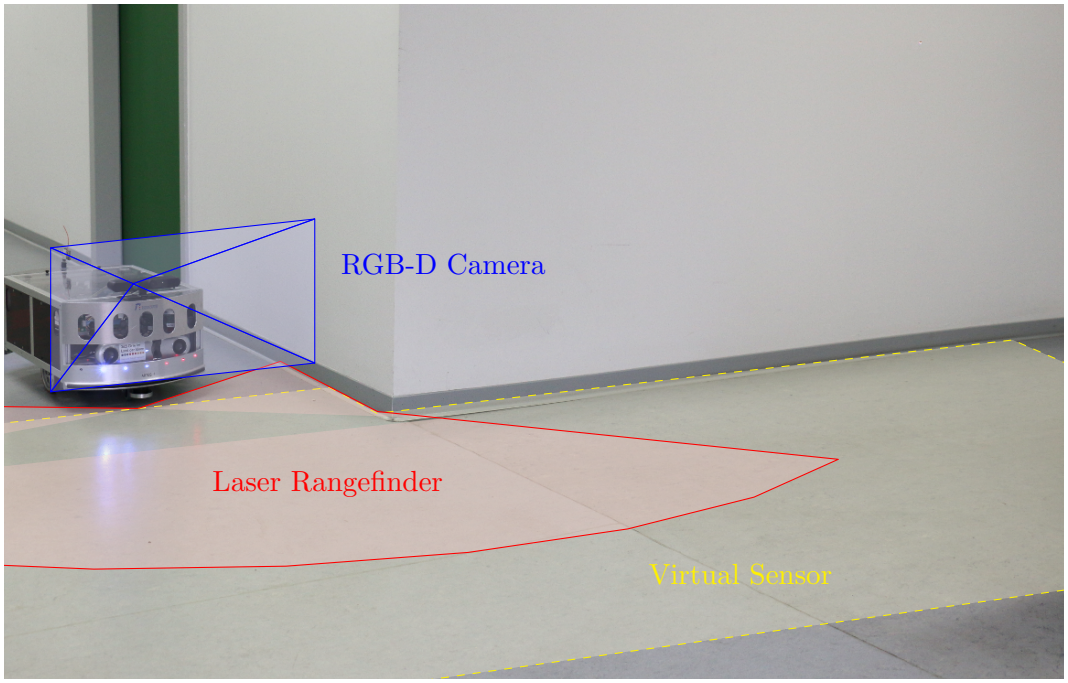


Figure 5.1.: Visualization of the local sensors and the virtual sensor of the robot Artos. The indicated ranges are for the visualization of the idea only and are *not* to scale.

5.1.1. Relation to Previous Works

What is further contained in this section differs significantly from the two previous works ([Arndt 11a] and [Arndt 12b]) in the way the data is used. In those works, the virtual obstacle avoidance sensor was implemented with the help of the iB2C integrated behavior-based control architecture (see [Proetzsch 10]). These previous works and implementations solely relied on the (single-target) tracking of people (as

described previously in section 4.6). The output-belief of the tracker was used as the input to a behavior-based virtual obstacle avoidance sensor. While this method did in fact work well in the experimental studies of these works, they were bound to the same assumptions as the employed trackers. Namely, the use of a single-target tracker requires a maximum of one person being present in the environment[3]. The concept could have been extended by a robust multi-target tracker and a fusion of all single beliefs to an input to the obstacle avoidance sensor, but this idea was not further pursued, because *robust* multi-target tracking with an exact estimation of the number of targets has proven to be very difficult (see section 4.7) for the targeted sensors (simple, anonymous, non-identifying passive infrared sensors).

Instead, the idea of employing a strict *tracking* of the people in the environment has been abandoned in favor of a more *pessimistic* estimation of the state of the environment. The tracking provides an *optimistic* view on the state of the environment, i.e. it tends to mark areas as "free" if the tracked subject is registered at a different position. As long as the estimated number of targets perfectly matches the number of people in the environment *and* all trackers correctly track their subject, this optimistic tracking is not incorrect. In a practical multi-target tracking context, this can however not be guaranteed and the chances are high, that at least one tracker provides wrong results. This may lead to the situation that some space is marked "free" although the real state is either "unknown" or worse, "occupied". For a safety-critical application, as given in this example, this is of course a situation that cannot be easily accepted.

For this reason, the more pessimistic concept of occupancy graph mapping (as introduced in section 4.8) has been used instead to determine the state of the environment. The occupancy graph mapping is by nature independent of the number of targets in the environment and can be used in a way that classifies all unknown space as "occupied" and only clearly unoccupied space as "free".

5.1.2. Relevant Particles on the Occupancy Graph

When using the occupancy graph to implement a virtual obstacle avoidance sensor, one of the most simple ideas is to take the path planned by the global path planner and overlay it on the occupancy graph. By looking at the intersection of that path with particles in proximity to the robot, obstacles can be detected. This technique can also look "around the corner" along the to-be-traversed edges. This principle is visualized in figure 5.2.

Unfortunately, it can be observed that only considering the particles directly *on* the planned path is not sufficient, as the particles (and thus the danger associated with them) may "hide" on edges close to the planned path (see the example in the figure). As dynamic obstacles (such as people) may traverse from such a hidden edge directly onto the path at any time, they cannot be neglected. To be able to maintain a safe speed at all times, more than just the actually planned path needs to be analyzed.

[3]The case of zero people in environment is handled implicitly by the tracker.

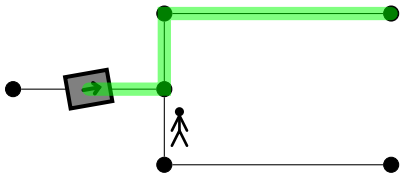


Figure 5.2.: Occupancy graph with obstacle and overlaid path to be traversed.

In fact, all *simple paths*[4]from the robot position in direction of travel within a maximum distance d_{stop} need to be checked for obstacles/particles. This length d_{stop} is equal to the distance the robot needs to come to a full stop. It is dependent on the braking deceleration of the robot and its current speed. This distance can be computed according to (5.1) using the current linear velocity v of the robot and its braking deceleration a_{brake}. For a derivation of that equation, please see section B.2 on page 165.

$$d_{\text{stop}} = \frac{v^2}{2 \cdot a_{\text{brake}}} \tag{5.1}$$

This procedure is depicted in figure 5.3. There are two simple paths in the direction of the robot (with a limited length). Both need to be analyzed for obstacles.

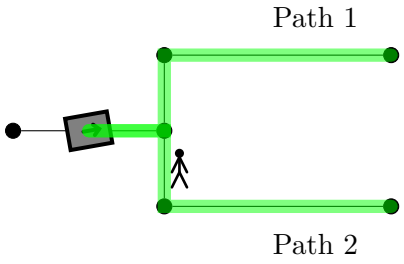


Figure 5.3.: Occupancy graph with obstacle and overlaid simple paths originating from the current robot position in the movement direction.

Of course, (5.1) is only valid if the obstacles themselves are static. Dynamic obstacles that move with a certain velocity through the environment (like e.g. people) may require larger stopping distances, as it is impossible for a moving object to stop instantaneously. If a dynamic obstacle is moving in the polar opposite direction as the robot, this effect is most prominent. Solving this issue exactly is difficult, as the velocities and headings of dynamic obstacles are usually unknown. One way to cope

[4]A simple path has no repeating vertices, i.e. if a vertex is visited more than once, the path does not qualify as a simple path.

with this is to choose the braking deceleration a_{brake} smaller than what the robot is really capable of, so that the safety margin is large enough.

The algorithm given as pseudo code in algorithm 5.1 is used to calculate the safe velocity v_{safe}. The algorithm takes a description of the robot's state in form of its current pose $(x, y, z, \alpha, \beta, \gamma)$ and its current twist[5] $(\dot{x}, \dot{y}, \dot{z}, \dot{\alpha}, \dot{\beta}, \dot{\gamma})$. The robot is parameterized by its assumed braking deceleration a_{brake}. The state of the environment is described using an occupancy graph consisting of the graph structure $G(V, E)$ and the particles with associated weights $P \subset (E \times \mathbb{R} \times \mathbb{R})$. In the following text, particles are called *candidate* particles, if they lie within the stopping distance of the robot and are not excluded due to some other reasons[6]. A particle is called *relevant*, if it additionally has a weight that is greater or equal than some threshold.

The algorithm first computes the distance the robot needs to come to a complete stop, using the current linear velocity $\dot{x}$ and the brake deceleration a_{brake} in line 1. This is the implementation of (5.1). In the two following lines, the robot position is mapped to a position on the graph and back to a Cartesian position, so that the robot is "fixed" to the graph. This becomes important shortly in section 5.1.3, where the influence of the local sensor systems of the robot is considered. The variable d_{smallest} (initialized in line 13) is used to keep track of the smallest relevant distance to an obstacle. Initially, this is set to the calculated stopping distance d_{stop} plus an additional term d_ε that allows the robot to accelerate beyond its current speed[7]. This d_ε can either be a constant that is chosen large enough to not limit the acceleration of the robot too much or it can be computed according to (5.2) using the current linear velocity $\dot{x}$, the maximum permissible (positive) acceleration $a_{\max}$ of the robot and the period t_{cycle} in which the algorithm is called periodically:

$$d_\varepsilon = \frac{2 \cdot \dot{x} \cdot a_{\max} \cdot t_{\text{cycle}} + a_{\max}^2 \cdot t_{\text{cycle}}^2}{2 \cdot a_{\text{brake}}} \tag{5.2}$$

If d_ε is computed according to this formula, it exactly represents the additional distance the robot can travel when linearly accelerating with its maximum permissible acceleration for the duration of one execution cycle. For the proof, please refer to section B.3 on page 166.

In lines 4 to 12, source and destination vertex as well as the offset distance d_{offset} between robot position and the source vertex are determined by detecting whether the robot is moving "in edge direction"[8]or against. The loop starting at line 14 iterates

[5] The twist is defined as the first derivative of the pose with respect to time.

[6] Lying within local sensor range of the robot is such a reason, but more on that soon.

[7] Consider the situation $d_{\text{smallest}} = d_{\text{stop}}$. Then the safe velocity would be calculated as $v_{\text{safe}} = \sqrt{2 \cdot a_{\text{brake}} \cdot d_{\text{stop}}} = \sqrt{(2 \cdot a_{\text{brake}}) \cdot (\dot{x}^2/(2 \cdot a_{\text{brake}}))} = \sqrt{\dot{x}^2} = \dot{x}$, which implies that the robot cannot accelerate to any speed larger than its current one.

[8] The graph is undirected, thus there is no real direction of an edge. As an edge is however described using a pair of source and destination vertex, there *is* a direction that can be defined.

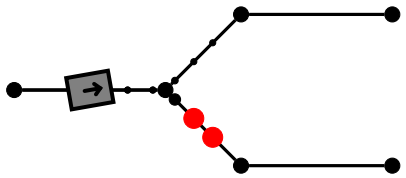
over all simple paths originating from the source vertex. The loop condition contains an optimization that does only loop over paths that are not too long. A path can be "cut off" at an edge, if that edge contains only positions that are not within the stopping distance of the robot.

The loop beginning in line 16 iterates over the edges (v_i, v_{i+1}) of the candidate path $(v_{\text{source}}, \ldots, v_n)$. With the help of the third loop in line 17, particles lying on that edge with a weight greater or equal than a threshold of 0.5 are searched. If such a particle is found and its position is close enough, so that the distance $d_{\text{robot-particle}}$ between the robot and the particle is smaller or equal to the currently smallest distance, then this distance is chosen as the smallest relevant distance.

The maximum permissible linear velocity v_{safe} of the robot can finally be computed in line 29 using the distance d_{smallest} and the brake deceleration a_{brake} of the robot according to (5.3). For a derivation of that equation, please see section B.2 on page 165.

$$v_{\text{safe}} = \sqrt{2 \cdot d_{\text{smallest}} \cdot a_{\text{brake}}} \tag{5.3}$$

The algorithm in its basic form, as given in algorithm 5.1 applied to two example scenarios is given in figure 5.4 and figure 5.5. In these figures, the robot pose is visualized using a rectangle with an arrow visualizing the yaw angle. Not all particles of the occupancy graph are drawn in the images. Only those, that are *candidates*, i.e. those that are within the distance d_{stop} relative to the robot are depicted. Their sizes are proportional to their weights. If a particle has enough weight to contribute to d_{smallest}, i.e. if it is *relevant*, it is visualized in red. In the first scenario (see figure 5.4), there are particles with enough weight to constitute an obstacle. The value of d_{smallest} and v_{safe} are in this case determined by the closest particle drawn in red.



$a_{\text{break}} = 0.25\text{m/s}^2$,
$v_{\text{current}} = 1\text{m/s}$,
$d_{\text{smallest}} = 1.53\text{m}$,
$v_{\text{safe}} = 0.8747\text{m/s}$

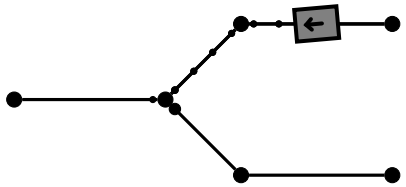
Figure 5.4.: Basic safe velocity calculation applied to an example scenario.

The second scenario (see figure 5.5), is different. While the state of the occupancy graph has not changed, the robot's state has. It has a different pose and a different v_{current} than before. In this configuration, there are no particles with a sufficiently large weight in range of the robot. Thus, the value of v_{safe} is calculated with the hypothetical distance of $d_{\text{stop}} + d_{\varepsilon}$ (see (5.2)), using the maximum permissible acceleration $a_{\text{max}} = 0.25\text{m/s}^2$ of the robot and the cycle time $t_{\text{cycle}} = 0.2\text{s}$. The printed value of d_{smallest} indicates that there is no relevant particle present.

Algorithm 5.1: Computation of the safe velocity v_{safe}

Require: $a_{\text{brake}} > 0$ {*the braking deceleration*}
Require: $G(V, E)$ {*the occupancy graph structure*}
Require: $p_i \in P, w_i \in W$ {*the particles and weights representing the occupancy graph data*}
Require: $(x, y, z, \alpha, \beta, \gamma)$ {*the current robot pose*}
Require: $(\dot{x}, \dot{y}, \dot{z}, \dot{\alpha}, \dot{\beta}, \dot{\gamma})$ {*the current robot twist*}
Returns: v_{safe} {*Permissible linear velocity*}

$d_{\text{stop}} = \dot{x}^2/(2 \cdot a_{\text{brake}})$ {*calculate the stopping distance at current linear velocity*}
$((v_{r_1}, v_{r_2}), t_r) =$ **MapCartesianPositionToGraph**$(G(V, E), x, y)$ {*map the Cartesian position of the robot to a position on the graph: edge (v_{r_1}, v_{r_2}), linear interpolation parameter t_r*}
$(x, y) \leftarrow \boldsymbol{x}(e_{r_1 r_2}, t_r)$ {*convert the resulting graph position back to Cartesian using (4.10)*}
if **RobotMovingInEdgeDirection**$(\gamma, (v_{r_1}, v_{r_2}))$ **then**
    $v_{\text{source}} = v_{r_1}$
    $v_{\text{dest}} = v_{r_2}$
    $d_{\text{offset}} = t_r \cdot \|\boldsymbol{x}_{r_1} - \boldsymbol{x}_{r_2}\|$
else
    $v_{\text{source}} = v_{r_2}$
    $v_{\text{dest}} = v_{r_1}$
    $d_{\text{offset}} = (1 - t_r) \cdot \|\boldsymbol{x}_{r_1} - \boldsymbol{x}_{r_2}\|$
end if
$d_{\text{smallest}} \leftarrow d_{\text{stop}} + d_{\varepsilon}$
for all $(v_{\text{source}}, \ldots, v_{n-1}, v_n) \in$ **SimplePaths**(v_{source}) such that **PathLength**$((v_{\text{source}}, \ldots, v_{n-1})) < d_{\text{offset}} + d_{\text{stop}}$ **do**
    $d_{\text{traveled}} \leftarrow 0$
    for all $i \in [1, n-1]$ **do**
        for all $(e_{jk}, t_j, w_j) \in P$ **do**
            if $(v_j, v_k) = (v_i, v_{i+1}) \wedge w_j \geq 0.5$ **then**
                $d_{\text{particle}} = d_{\text{traveled}} + t_j \cdot \|\boldsymbol{x}_i - \boldsymbol{x}_{i+1}\|$
                $d_{\text{robot-particle}} = d_{\text{particle}} - d_{\text{offset}}$
                if $d_{\text{robot-particle}} > 0 \wedge d_{\text{robot-particle}} \leq d_{\text{stop}}$ **then**
                    $d_{\text{smallest}} \leftarrow \min\{d_{\text{smallest}}, d_{\text{robot-particle}}\}$
                end if
            end if
        end for
        $d_{\text{traveled}} \leftarrow d_{\text{traveled}} + \|\boldsymbol{x}_i - \boldsymbol{x}_{i+1}\|$
    end for
end for
$v_{\text{safe}} = \sqrt{2 \cdot d_{\text{smallest}} \cdot a_{\text{brake}}}$
return v_{safe}



$a_{\text{break}} = 0.25\text{m/s}^2$,
$v_{\text{current}} = 1.2\text{m/s}$,
$d_{\text{smallest}} = \infty\text{m}$,
$v_{\text{safe}} = 1.25\text{m/s}$

Figure 5.5.: Basic safe velocity calculation applied to an example scenario with no obstacles in range.

5.1.3. Local Sensor Systems

If the safe velocity, as determined by this method, is directly used to constrain the movement of the robot, problems may arise. This is due to the fact that the occupancy graph is *pessimistic*, i.e. that it only clearly identifies *un*occupied areas. In occupied areas, the chance for false-positive results is very high. The safe velocity is thus often estimated to be too low, which can easily lead to the situation where the robot moves very slowly or is unable to move at all.

This issue can be overcome by incorporating knowledge about the *local* sensor systems in the process of determining the candidate particles and thus the safe velocity. To do so, the approximate local sensor range of the robot is estimated using a polygon which is transformed according to the current robot pose on the graph. If a candidate particle does now fall into the polygon of the local sensor range, it is *not* considered for calculating d_{smallest} and thus v_{safe}, i.e. it cannot be a relevant particle. This practice is justified by the assumption that the local sensors provide more precise measurements than what is contained in the occupancy graph. Additionally, the existing local path planner is assumed to cope with obstacles that may be present within local sensor range. The idea is depicted in figure 5.6. While there are candidate particles outside of the local sensor outline, all candidates inside have been removed.

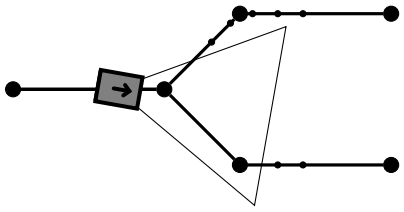


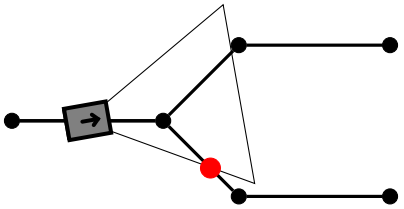
Figure 5.6.: Removal of candidate particles that lie within the range of the local sensor systems of the robot.

The changes to algorithm 5.1 to exclude particles that lie within the range of the local sensors are trivial. In line 21, where the decision whether a particle is relevant is made, a predicate that checks for exclusion in the local sensor outline must be added. This predicate, $\neg$**ParticleWithinLocalSensorRange**$(p_j, (x, y, z, \alpha, \beta, \gamma))$, needs to

be added with a logical "and" operator ($\wedge$) to the expression, so that the statement becomes:

$$\textbf{if } d_{\text{robot-particle}} > 0 \wedge d_{\text{robot-particle}} \leq d_{\text{stop}} \\ \wedge \neg \textbf{ParticleWithinLocalSensorRange}(p_j, (x, y, z, \alpha, \beta, \gamma)) \textbf{ then}$$

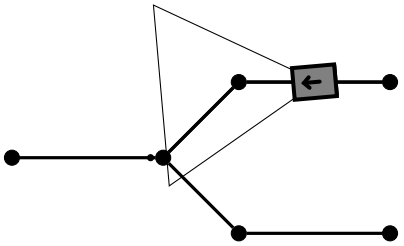
If the same state of the environment and the robot as from figure 5.4 is taken, but this time *with* handling of the local sensor range, the situation in figure 5.7 results. The local sensor range of the robot is represented by the polygon in front of the robot. By comparing this figure carefully with figure 5.4, it can be seen that most of the candidate particles are removed by the local sensor outline. The one particle that is left exceeds the threshold and thus determines the values of d_{smallest} and v_{safe}.



$a_{\text{break}} = 0.25\text{m/s}^2$,
$v_{\text{current}} = 1\text{m/s}$,
$d_{\text{smallest}} = 1.884\text{m}$,
$v_{\text{safe}} = 0.9705\text{m/s}$

Figure 5.7.: Safe velocity calculation with consideration of the local sensor system of the robot, applied to an example scenario.

The analog situation to figure 5.5 is shown in figure 5.8. It can also be seen in this figure, that the candidate particles are removed. However, the values of d_{smallest} and v_{safe} do not change, as there were no obstacles in the first situation and thus considering the local sensor outline does not change the assessment of the situation.



$a_{\text{break}} = 0.25\text{m/s}^2$,
$v_{\text{current}} = 1.2\text{m/s}$,
$d_{\text{smallest}} = \infty\text{m}$,
$v_{\text{safe}} = 1.25\text{m/s}$

Figure 5.8.: Safe velocity calculation with consideration of the local sensor system of the robot, applied to an example scenario with no obstacles in range

5.1.4. Experiments

The effectivity of the general idea of virtual obstacle avoidance sensors has been evaluated in the previous works of the author ([Arndt 11a, Arndt 12c]). Instead of

simply replicating the same results in this work, this experimental section makes use of the newly introduced virtual obstacle avoidance sensors based on the occupancy graph mapping. This is in contrast to the previous works, which solely relied on (single) target tracking for state estimation.

The experiments for this thesis were mostly conducted in the simulation environment (see section A.4 for a description of the simulation environment) in order to be able to execute a large, statistically relevant number of experimental runs. To show the applicability beyond simulation and the validity of the simulation results, a smaller number of experiments has also been conducted in reality.

All experiments were set in the office environment of the Robotics Research Lab (see section A.3). In each of the runs, the mobile robot Artos (see section A.2) was driving either from place A to B or from place B to A (the locations of the places are indicated in figure A.7). Additionally, a human being was walking in the environment. In the simulation experiments its trajectory was predefined, for the experiments in reality it was random. The experiments have been parameterized by the set of sensor nodes, which were activate during the runs. A special case is the situation in which *none* of the nodes have been activated. This case is equivalent to the situation in which no external sensor information at all is available. The robot thus runs in "conventional" mode for these experiment runs, i.e. it behaves as if it cannot "see" further than what its local sensors are able to perceive. Experiment class 1 does represent this "conventional" mode, as in this class, not a single sensor node was active. Class 2 does only make use of the nodes with identifiers 2, 7 and 9. Class 3 is the opposite of class 1 – in this class, all nodes are *en*abled. For each experiment class, a certain number of experiment runs have been conducted. For these runs, minimum, maximum, arithmetic mean and standard deviation σ have been computed. For a small number of runs with the real robot in the real environment, table 5.1 shows the results.

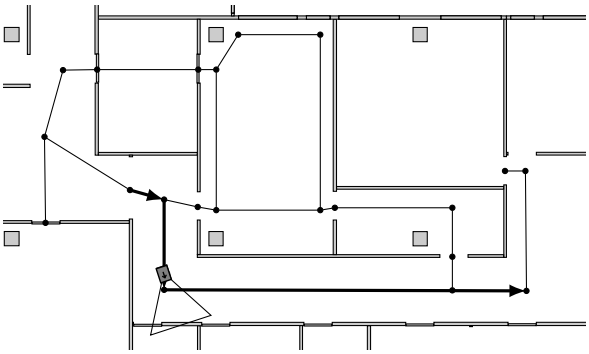
Class	**Action**	**Runs**	**Min** [s]	**Mean** [s]	**Max** [s]	σ [s]
1	$A \rightarrow B$	4	59.6	60.6	61.8	0.81
	$A \leftarrow B$	3	58.8	61.1	65.4	3.02
2	$A \rightarrow B$	3	59.1	60.4	62.2	1.33
	$A \leftarrow B$	4	58.1	60.5	64.2	2.39
3	$A \rightarrow B$	3	57.9	58.3	58.9	0.42
	$A \leftarrow B$	3	56.8	58.2	59.4	1.07

Table 5.1.: Data of experimental results with the real robot.

The table shows improvements in the mean times the tasks needed to complete when enabling more sensor nodes (from class 1 to class 2 and from class 2 to class 3). The table also shows that there is no significant difference in time if the path is traversed

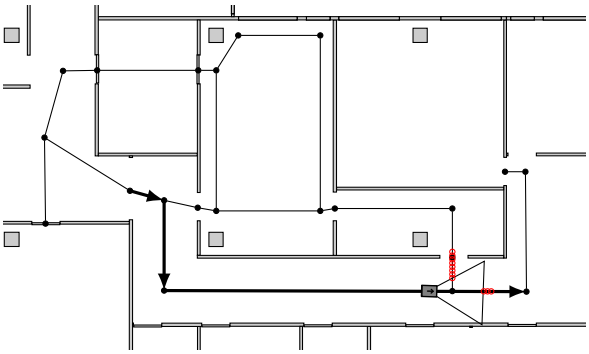
from A to B or the other way round. For this reason, the simulation experiments that are discussed soon, have only been conducted in one direction ($A \rightarrow B$).

To give the reader a more vivid view on the experiments in reality, one of the runs is visualized in figure 5.9. In this figure, also the local sensor outline of the robot and the relevant particles are visualized, similar as in the artificial examples at the beginning of this section. There are two aspects that can be seen in this figure. In



(a) $v_{\max} = 0.40\text{m/s}$, $v = 0.33\text{m/s}$. The speed is within the bounds, the robot may accelerate.

(b) Corresponding camera image.



(c) $v_{\max} = 0.37\text{m/s}$, $v = 0.50\text{m/s}$. The robot has to reduce its speed.

(d) Corresponding camera image.

Figure 5.9.: Visualization of two situations during an experiment run that demonstrate the virtual obstacle avoidance sensor using an occupancy graph.

figure 5.9a, there *are* particles which could be relevant[9] in front of the robot, but its local sensor systems confirm that there is no obstacle present in the driving direction. It can also be seen that the permissible speed $v_{\max}$ is larger than the current speed v, which allows the robot to accelerate. In figure 5.9c, the situation is different. In this case, there exist relevant particles which cannot be observed by the local sensors of the robot. Those are particles that lie either directly beyond the end of the (reliable)

[9] In this example run, all sensor nodes were inactive, thus *all* particles on the graph which are candidate particles are also relevant particles.

local sensor range (the ones in driving direction) or to the side of the robot, where the local sensor coverage is particularly bad. Those particles to the left of the robot are the closest ones, thus they define the safe speed which is only 0.37m/s at that situation. As the robot is actually driving faster than that speed, it must decelerate. This deceleration was clearly visible while observing the experiment run. Another, more thorough visualization of an experiment run with the real robot is given in the appendix in section C.3.

As already mentioned, a much larger amount of experiments has been conducted using the simulation environment, to be able to better quantify the reduction of time. These results are given in two forms. First, table 5.2 shows s similar representation as given for the runs in reality in table 5.1. Additionally, all the raw data points for all three classes are plotted as a scatterplot in figure 5.10.

Class	**Action**	**Runs**	**Min** [s]	**Mean** [s]	**Max** [s]	σ [s]
1	$A \rightarrow B$	100	63.1	65.1	74.6	2.13
2	$A \rightarrow B$	99	59.8	62.4	73.7	2.46
3	$A \rightarrow B$	100	57.1	58.9	67.3	1.45

Table 5.2.: Data of experimental results in the simulation environment.

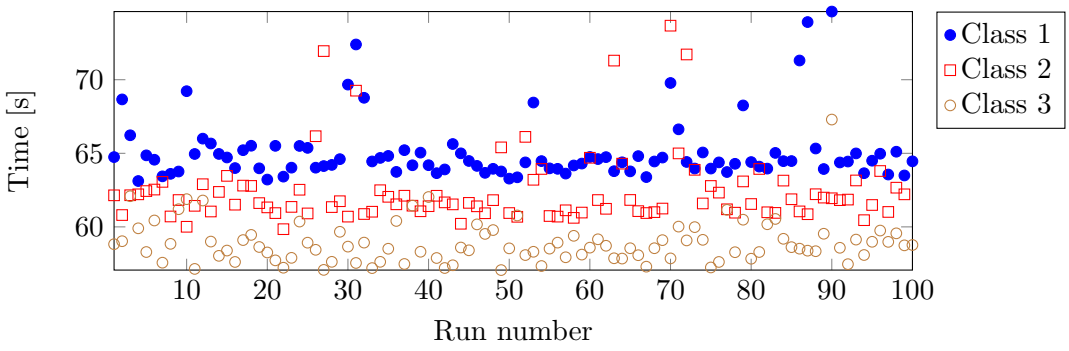


Figure 5.10.: Scatterplot of the experimental results in the simulation environment.

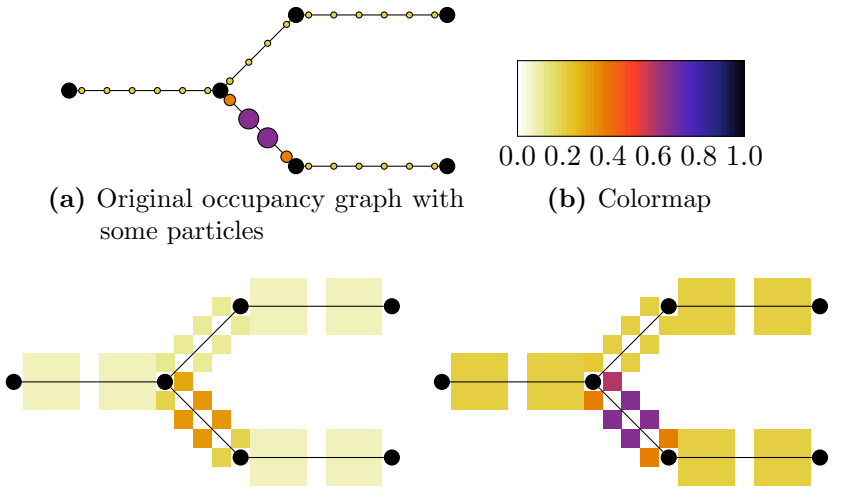
By comparing the table of the simulation results with the runs using the real system, the same effect as described above can be observed. The more sensor nodes are active and deliver valid data, the smaller the time to complete the task is. Interestingly, the effect is even more pronounced in the simulation environment, as can be seen when comparing classes 1 and 2 in reality and simulation. In the case of all nodes active (class 3) the mean times are almost identical between reality and simulation. The scatterplot shows a few outliers, especially in classes 1 and 2. However, these are most likely the result of a general navigation issue of the robot and are probably not

caused by the virtual obstacle avoidance sensor. The majority of data points shows the same improvements that are also indicated by the mean times given in the table.

5.1.5. Application to Occupancy Grids

As occupancy grids are very popular in robotics in general (see section 2.1.1.1) this section considers the integration of the data if an occupancy grid map is used for obstacle avoidance. It is important to notice that this section only provides the general ideas of how to implement the virtual obstacle avoidance sensors with standard occupancy grids. There are explicitly no experiments or further evaluations.

To use the virtual obstacle avoidance with a robot's control entity that can only handle occupancy *grids* and not occupancy graphs directly, the particles of the occupancy *graph* can be mapped onto an occupancy *grid*, as discussed before in section 4.3.2. This process is exemplarily visualized in figure 5.11. While figure 5.11a shows the original



(a) Original occupancy graph with some particles

(b) Colormap

(c) Occupancy grid generated with proportional mapping of particle weights to grid cells.

(d) Occupancy grid generated with full mapping of the particle weights to grid cells.

Figure 5.11.: Mapping the occupancy graph to an occupancy grid.

graph structure with the particles that define the current state of the occupancy graph, figure 5.11c shows the same data mapped to an occupancy grid using the "proportional" mapping method while figure 5.11d uses the "full" mapping method. For the mapping process, an edge width of 0.5m has been set. The grid structure has a resolution of 0.25m in horizontal as well as in vertical direction. The colormap in figure 5.11b indicates how the colors can be mapped to weights/probabilities. By comparing the two mappings, it can be seen that for the obstacles avoidance using an occupancy grid, the "full" mapping should be used, as this mapping guarantees

that every grid cell is assigned the full weights of the particles. The "proportional" mapping spreads the weight of a particle over (possible) several cells and thus the threshold for a cell to be considered occupied might not be reached.

5.2. Risk-Reduced Path Planning

It is a general requirement that robots should always be safe – or as Isaac Asimov wrote down in his famous first law of robotics [Asimov 13, p. 44]:

> "A robot may not injure a human being, or, through inaction, allow a human being to come to harm."
>
> First of the Three Laws of Robotics, Isaac Asimov

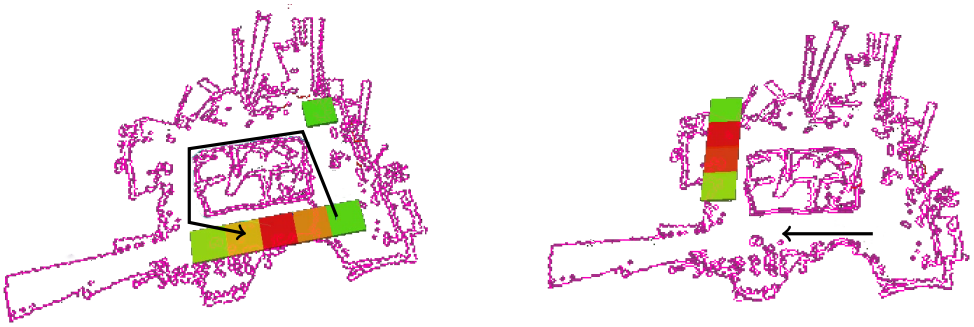
For mobile robots, this is usually achieved using local path planning and obstacle avoidance to not hit human beings which constitute a dynamic obstacle for the mobile robot. This is what has also been introduced previously in section 5.1.

However, the mere presence of a robot, even if it is not moving, may present a hazard to people in the environment, as someone might e.g. oversee it and trip over it. This gives birth to the idea to optimize the *global* path planning in a way that the chance for a collision, i.e. the probability of meeting human beings, during execution of a task is reduced before even starting to traverse the path.

5.2.1. Relation to Previous Works

The general idea of the applications contained in this section has been previously published in the proceedings of the joint conference of ISR 2014 and ROBOTIK 2014 [Arndt 14]. The basic idea is to make use of information from external sensors not only when traversing a path with a mobile robot, but already in the planning phase. This is related to the field of human aware motion planning. Instead of only avoiding collisions when they are imminent, paths can be planned in a way to avoid risky situations right away. Also a still-standing robot (due to collision avoidance) could be a danger to a dynamic obstacle, such as a human.

The very basic idea is to use the same principle as in the previous application of virtual obstacle avoidance sensors (see section 5.1) and to consider regions of high probability of a person being present as "blocked" in the global path planning process. Exactly this approach has been evaluated in some experiments done at the Laboratório de Automação e Robótica (LARA) at the University of Brasília, Brazil, see [Arndt 14, 4.2.1]. Two situations of this experiment are shown in figure 5.12. In one situation, the direct path from start to goal of the robot is "blocked" due to the external sensor information. In the other one, it is free. Of course, the practical relevance of this "black and white" view on the situation is low, as it may lead to very long detours of the robot or even to the impossibility to find a path at all.



(a) Robot needs to take the long path as the direct path is blocked.

(b) The robot can take the direct path to the goal

Figure 5.12.: Visualization of planned paths, depending on the state of the environment. Originally published in [Arndt 14].

The rest of this section is similarly structured as [Arndt 14], but it aims to be more thorough and explain the ideas and approaches more in detail. After introducing related works in the field, a method for quantifying and evaluating risk is given. This is followed by the algorithm that has been implemented for mobile robots to achieve risk-reduced path planning. Finally, experimental results of the application of the planner are presented.

5.2.2. Human Aware Planning

Sisbot et al. have coined the term *Human Aware Motion Planner (HAMP)* in [Sisbot 07]. In a previous work, [Sisbot 05] motivates the need for robot movement that is acceptable according to social standards and legible for humans in the same environment. They defined three criterions to achieve this kind of robot movement. The first one is the *safety criterion* which is based on the distance between the robot and the human. The further away a robot is, the safer the human will feel. The authors also note that this "safe distance" is a subjective value that varies between different people and is also dependent on their current state, i.e. whether they are sitting down or standing. The second criterion, the *visibility criterion*, states that a human feels more safe and comfortable if the robot is visible in his field of view and the human is thus able to observe the robot and possibly being able to understand what its intentions are. The *hidden zones criterion* is an extension that aims to avoid robot positions that are hidden with respect to the human. This may e.g. be the case if an obstacle in the environment is blocking the direct view from human to robot. Such positions are penalized in order to avoid to surprise the human if the robot is suddenly entering the field of vision. While [Sisbot 05] only describes simulation experiments, [Sisbot 07] also contains the documentation of real-world experiments. The authors

acknowledge "significantly different" results with their human aware planner compared to the classic approach, however both of the works miss a measurement of "human awareness" or safety.

Alami et. al [Alami 06] highlight the research challenges of the two research projects *Physical Human Robot Interaction in Anthropic Domains (PHRIDOM)* and *Physical Human-Robot Interaction: Dependability and Safety (PHRIENDS)*. It is relevant in various ways, as it acknowledges the need for metrics of safety and dependability [Alami 06] and states that "Safety and dependability are the keys to a successful introduction of robots into human environments." [Alami 06, I.]. The work mainly focuses on robotic manipulators and their design from mechanics to software to achieve safety, but the authors also draft the need for safety and dependability in unstructured, "everyday" environments [Alami 06, IV. B.]. The work explicitly names the *Abbreviated Injury Scale (AIS)* as an example metric for quantifying safety of (automotive) systems through the amount of injury they may cause to human beings [Alami 06, II. C].

Cirillo et al. describe a human-aware planner that employs *interaction constraints* to create high-level plans for mobile robots in environments together with human beings [Cirillo 10, Cirillo 12]. Depending on these interaction constraints, these plans could e.g. focus on safety or comfort of the human beings and on the task themselves. The authors name an example scenario where these constraints can be used to have a cleaning robot which is never in the same room as the person [Cirillo 10, 4.1]. Similar to the ideas of this thesis, the authors also explicitly use information from external sensors, that are present in the (smart) environment to determine the current state of the environment and to predict future situations [Cirillo 12, 1, 5.1, 5.3].

5.2.3. Risk and its Quantification

First of all, when speaking of "risk-reduced" path planning, a definition and more importantly a quantification of risk must be given. In the Stanford Encyclopedia of Philosophy, Sven Ove Hansson states that risk refers "often rather vaguely, to situations in which it is possible but not certain that some undesirable event will occur" [Hansson 14, 1.]. Such undesirable events could in the example of this work e.g. be a collision between a person and a robot or also just some other inconvenience caused by the robot. With regard to quantification of risk, the definitions of risk as "the probability of an unwanted event which may or may not occur" [Hansson 14, 1. (3)] or "the statistical expectation value of an unwanted event which may or may not occur" [Hansson 14, 1. (4)] are also highly important.

In the following, the term *risk* is used in the quantitative sense like the latter definition given by Hansson. A high risk means that there is a high probability of something going wrong, i.e. a collision or accident occurring, while a risk of zero represents the situation in which the robot is absolutely safe for the human beings in the environment.

In the following, the *risk* is quantified by a *risk cost* function with a codomain equal to the interval $[0, 1]$. There are various candidates for the input arguments of this

function, that refer to the states of people and robots in the environment, but in the following only the distance between the robot and the (closest) person is used. An additional argument could e.g. be the current speed of the robot relative to the person's speed, to express the statement that a fast moving robot might be more dangerous than a slowly moving robot.

The proposed cost function is given in (5.4). It has been originally published in [Arndt 14, 3.1]. It is a piecewise polynomial function that needs to be parameterized using the parameters a and b.

$$f_{a,b} : [0, \infty] \to [0, 1]$$
$$f_{a,b}(d) = \begin{cases} 1 - (a^{-1}d)^b & d \leq a \\ 0 & d > a. \end{cases} \tag{5.4}$$

These parameters control the shape of the function and must be adapted to suit the scenario. Parameter a describes the "safe distance". From this distance on, the cost will always evaluate to zero, meaning the situation is evaluated to be perfectly safe. The value must be greater than zero ($a > 0$). The second parameter, b, describes the degree of the polynomial. It must also be greater than zero ($b > 0$) and defines the steepness of the curve. See figure 5.13 for a visualization of the function for different parameters.

A very important aspect of that function is the fact that it is monotonically decreasing for $a > 0$ and $b > 0$. The proof can be found in the appendix in section B.4. This fact is used later to optimize the algorithms that are used to apply this cost function to a graph structure.

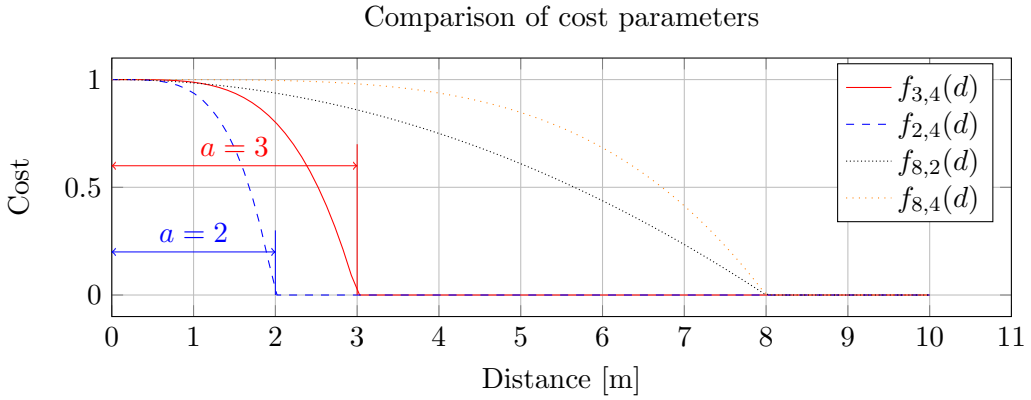


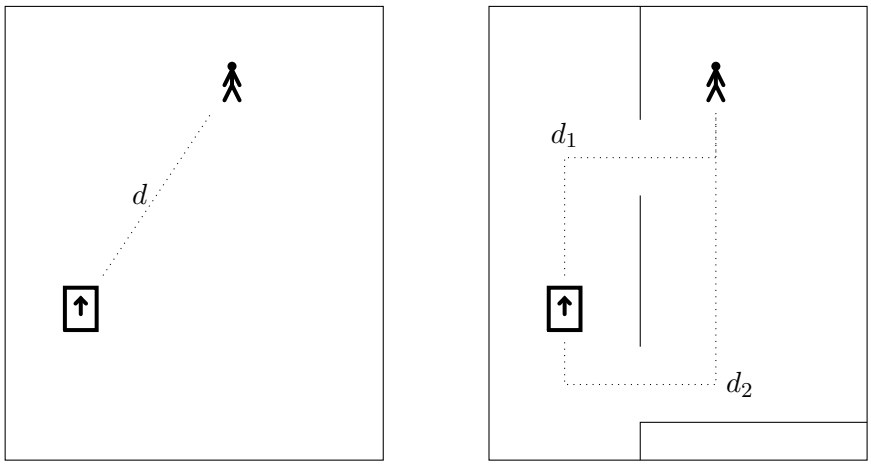
Figure 5.13.: Plot of the cost function for different pairs of parameters, similarly published in [Arndt 14].

The choice of the two parameters a and b usually depends on the dynamic properties of the robot and the type of the environment, as well as the constraints it puts on the

robots and people. If the robot can drive very fast, it is advisable to choose a larger value of a and maybe also a smaller value of b so that the fact that the robot can change its position very fast is accounted for. During experiments in an indoor office environment, values of $a = 8$ and $b = 2$ have shown to be applicable and to provide good results.

The cost function (5.4) needs to be applied to the shortest distance $d_{\min}$ between the robot and a person in the environment. Without loss of generality, only a single person is assumed to be present[10]. In a completely unconstrained environment, the distance can trivially be computed by applying the Euclidean norm to the vector between robot and human, see figure 5.14a for a depiction of the situation.

In practice however, this simple situation is rare, as movement in typical environments is usually constrained by fixed obstacles such as walls that cannot be overcome by either robots or humans. In such situations, the shortest path between robot and person has to be found. The length of this shortest path $d_{\min}$ is then the value for which the cost function needs to be evaluated. This situation is depicted in figure 5.14b with d_1 being the shortest path ($d_{\min} = d_1$).



(a) Open space **(b)** Constrained by walls

Figure 5.14.: Determination of the distance d for the risk cost function in trivial open space and in a typical, constrained environment.

In the general case, a robot would have to consider many different paths between its current position and the position of the human to find the shortest path and thus the risk associated with the current situation. Luckily, the graph on which the people are tracked and which has been used for occupancy graph mapping (see section 4.3) gives

[10] If there are multiple, the procedure can be carried out for each of them and the minimum distance $d_{\min}$ can be taken to calculate the risk costs. Other methods of fusion may also be applicable, later, a way to apply the cost function to a probabilistic description of the state of the environment is introduced.

a very good representation of the paths on which the people, but also on which the robot can move. The same graph has thus also been used for path planning, which implies that the possible paths robots and humans can take are identical.

A very naive approach to finding the safest path from start A to goal B with the human residing at position C could be as follows:

- For each existing candidate path from start A to goal B, simulate the traversal of the path by the robot.
- For each position on the path, calculate the cost by finding the shortest distance $d_{\min}$ between the currently simulated robot position and the human position C. Evaluate the safety cost function at that point: $f_{a,b}(d_{\min})$ to obtain a value describing the safety of that position.
- Calculate the mean safety cost of the path.
- At the end: Choose the path with least cost, as it is the safest one, according to the metric.

While this clearly illustrates the principle, it is not very elegant and efficient after all[11]. Instead of finding the shortest path between the human and the robot and afterwards evaluating the safety of that robot position, a different approach is used. For the sake of understandability and clarity, at first, a deterministic position of the person is assumed, i.e. the position is known and not uncertain. Later, the approach is extended to the probabilistic case in which the position is *not* exactly known.

The goal is to create a *safety cost graph* that is structurally identical to the graph which describes the environment, but whose edge weights represent the cost function applied to the position of the person. Coming back to the example, figure 5.15 depicts such a situation. The cost function has been evaluated on the edges of the graph and the costs for traversing each edge are written next to the edges.

The edge weights are computed by "flooding" the cost onto the graph, originating from the position of the person. Vividly, consider pouring a paint bucket at the position of the person onto the graph – as the paint spreads along the edges of the graph, so does the risk cost. In this thesis, this process is implemented using the recursive procedure **ApplyToEdge** which is given in algorithm 5.2[12]. This procedure is initially called with the position of the person (e_{ij}, t) to start the recursion. Just to recap, this means the person is located on the edge between vertices v_i and v_j with $t \in [0, 1]$ being the parameter for linearly interpolating between the two.

Line 1 starts the recursion with the visited set V set to the empty set and the distance traveled d to zero. If an edge has already been visited in the current call stack, the recursion ends in line 4. Thanks to the requirement of having a monotonic cost function ($a > b \Rightarrow f(a) \leq f(b)$), the procedure can terminate the internal recursion as

[11] However, this approach is revisited in the next section, when predictions about the evolution of the state are made.

[12] Please note that the original publication in [Arndt 14] unfortunately contained an error (V was a global variable), now it is a parameter of the procedure.

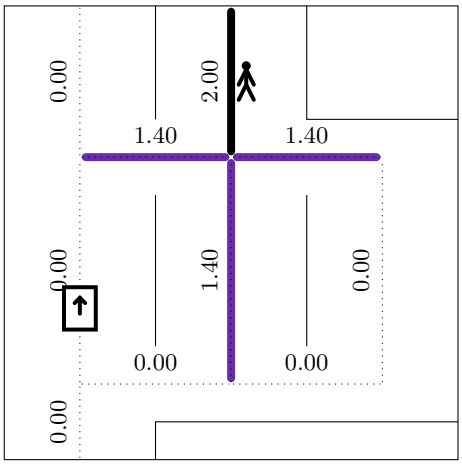


Figure 5.15.: Applying the cost function to a graph.

soon as one distance evaluates to zero in line 7. If the recursion did not terminate so far, in lines 9 to 12 the length of the edge segments behind and in front of the current value of t, along with the costs corresponding to the lengths are calculated. In line 13, these costs are summed to the current weight of the edge. The recursion is continued in lines 16 and 21 for the successors and the predecessors of the current edge e_{ij}. Please note that the predecessors are only evaluated in the initial situation if $t \neq 0$, in all other cases, the recursion is only done in "forward" direction.

In the actual implementation, the integrals in lines 10 and 12 might be available in a closed form, depending on the cost function used. For the previously proposed function (5.4), the integral is given in (5.6). To increase readability, the symbol of the distance has been changed from d to x in the formula. The proof for this closed form is available in the appendix in section B.5.

$$F_{a,b} : [0, a] \to \mathbb{R}$$

$$F_{a,b}(x) = \int f_{a,b}(x)\,\mathrm{d}x \tag{5.5}$$

$$= x - \frac{1}{a^b \cdot (b+1)} x^{b+1}. \tag{5.6}$$

What this procedure formally does is to convolute the safety cost function with the Dirac delta function[13] shifted to the position of the person on the graph. With this in

[13]The Dirac delta function $\delta(x)$ is zero anywhere except at $x = 0$. Over its whole domain, it has an integral of 1. It can therefore be regarded as some kind of degenerate "probability density function" of a deterministic (i.e. non-random) event. A function convoluted with the Dirac delta function results in the original function.

Algorithm 5.2: Mapping the cost function onto the graph, similarly published in [Arndt 14].

Require: $\forall i, j : \|\boldsymbol{x}_\mathbf{i} - \boldsymbol{x}_\mathbf{j}\| > 0$
Require: $a > b \Rightarrow f(a) \leq f(b)$
1: **ApplyToEdge**$(e_{ij}, t, 0, \emptyset)$ {*Start the recursion*}
2: **ApplyToEdge**(e_{ij}, t, d, V) :
3: **if** $e_{ij} \in V$ **then**
4: **return** {*edge already visited, we may stop*}
5: **end if**
6: **if** $f(d) = 0$ **then**
7: **return** {*cost evaluates to zero, we may stop*}
8: **end if**
9: $l_a = t \cdot \|\boldsymbol{x}_\mathbf{i} - \boldsymbol{x}_\mathbf{j}\|$ {*the length of the edge segment "behind"*}
10: $c_a = \int_d^{d+l_a} f(x)\,\mathrm{d}x$
11: $l_b = (1-t) \cdot \|\boldsymbol{x}_\mathbf{i} - \boldsymbol{x}_\mathbf{j}\|$ {*the length of the edge segment "in front"*}
12: $c_b = \int_d^{d+l_b} f(x)\,\mathrm{d}x$
13: $w_{e_{ij}} \leftarrow w_{e_{ij}} + c_a + c_b$ {*sum up cost to edge* e_{ij}}
14: **for all** $e_{i'j'} \in$ **OutEdges**(v_j) **do**
15: {*Recursively apply to all successors*}
16: **ApplyToEdge**$(e_{i',j'}, 0, d + l_b, V \cup \{e_{ij}\})$
17: **end for**
18: **if** $t \neq 0$ **then**
19: **for all** $e_{i'j'} \in$ **OutEdges**(v_i) **do**
20: {*Recursively apply to all predecessors*}
21: **ApplyToEdge**$(e_{i',j'}, 0, d + l_a, V \cup \{e_{ij}\})$
22: **end for**
23: **end if**

mind, it is rather easy to generalize the method to the non-deterministic case, i.e. the *probabilistic case* of a human position. In this case, the position of the human is no longer exactly known, but it is represented by a distribution (a belief) which defines where the human *might* be located with some probability. For the belief defined as $\text{belief}(x_t) = p(x_t \,|\, z_{0\ldots t}, u_{0\ldots t})$, the convolution looks like this:

$$(f * \text{belief})(x) = \int_{-\infty}^{\infty} f(\tau)\,\text{belief}(x - \tau)\,d\tau \tag{5.7}$$

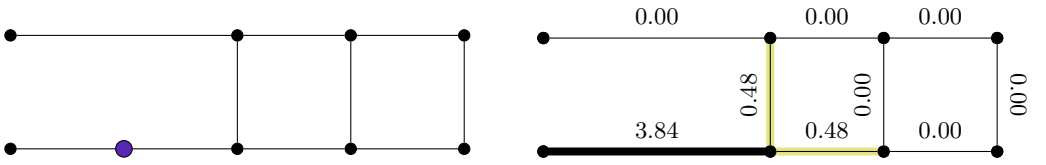
Solving this convolution in general might be hard, but fortunately, the belief throughout this work is represented using particles of a particle filter (see section 4.2.4). For this reason, it is possible to just use the same algorithm that was used for the deterministic case and apply it for every single particle. As the edge weights are accumulated (see line 13 of algorithm 5.2), this works out of the box. The process is formally described in algorithm 5.3.

One example of mapping the particle weight to the edges of a graph can be found

Algorithm 5.3: Mapping the cost function onto the graph in the probabilistic case with the belief being represented by particles. Originally published in [Arndt 14].

1: $\forall e_{ij} : w_{e_{ij}} \leftarrow 0$ {*Initialize all edge weights with zero*}
2: **for all** $(e_{ij}, t, w) \in P$ **do**
3: {*For each particle with edge* e_{ij} *and interpolation parameter* t}
4: **ApplyToEdge**$(e_{ij}, t, 0, \emptyset)$
5: **end for**

in figure 5.16. In this example, a single particle with a weight of 1 is mapped to the edges of the graph using the cost function $f_{3,4}$.



(a) Graph with particle position

(b) Graph with cost function mapped to the edges.

Figure 5.16.: Visualization of mapping the weight of a single particle (situated on the lower left edge) to the edges of the graph.

5.2.4. Fusion of Risk and Distance

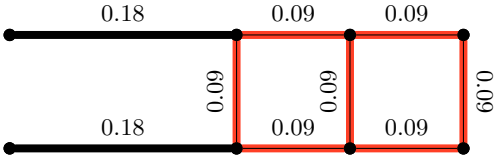
By now, a graph with edge weights representing the safety costs associated with traversing these edges has been created. Standard algorithms that find shortest paths on a graph (e.g. Dijkstra's algorithm) can be used at this point to find the safest path. However, the *safest* path is not always the desired path. Rather, in practice, a trade-off between a *safe* and a *short* (i.e. efficient) path needs to be found.

Planning using this trade-off is realized by fusing the safety-cost graph with a distance graph whose edge weights are proportional to the Euclidean lengths of the edges. The result of this fusion is another graph on which finally a shortest-path algorithm can be run to find *safe* and *short* paths.

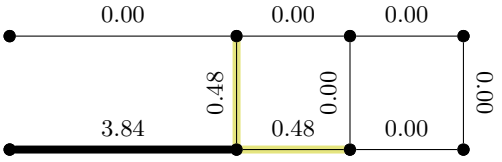
Before fusion, both graphs first need to be normalized so that their edge weights each sum up to a total value of 1 (except in the border case, where all weights of one graph are zero, in this case, no normalization is performed). This normalization is needed to make the fusion independent of the absolute Cartesian edge lengths and the parameters of the cost function. The formula that describes the fusion is given in (5.8), which has originally been published in [Arndt 14, 3.4]. The $w^{\star}_{e_{ij}}$ denotes the weight of the edge e_{ij}, for $\star = d$ in the normalized distance graph and for $\star = r$ in the normalized risk-cost graph.

$$w_{e_{ij}} = \alpha \cdot w^d_{e_{ij}} + \beta \cdot w^r_{e_{ij}} \tag{5.8}$$

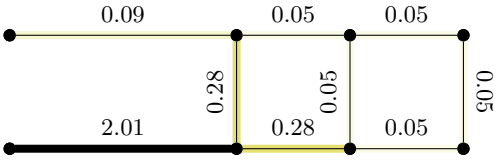
An example fusion of a distance graph with a cost graph can be found in figure 5.17. This example uses the cost graph that has been calculated to demonstrate the application of the cost function in figure 5.16. It is fused with the distance graph, the parameters for the fusion have been set to $\alpha = 0.5$ and $\beta = 0.5$.



(a) Distance graph



(b) Cost graph



(c) Result of the fusion

Figure 5.17.: Example for the fusion of a distance graph with a cost graph.

5.2.5. Experiments

Several experiments have been conducted to compare the risk-reduced path planning with the classical path planning approach that only operates on the distance graph. In principle, the input belief of the risk-reduced path planner can be connected to any state estimator that provides information about the whereabouts of the people in the environment. For this experiment, it has been connected to the output of the probability hypothesis density (PHD) filter described in section 4.7.4. This way, the risk-reduced path planning can also work with multiple targets, instead of only a single one, as the PHD filter is a multi-target tracker.

The experiments have mainly been conducted within the simulation environment SimVis3D (see section A.4). Just like in the previous experiment, the simulated robot

ARTOS was placed in the simulated office environment of the Robotics Research Lab, equipped with simulated AmICA nodes (the exact layout and sensor placement is described in section A.3). A simulated person was following a predefined trajectory to generate activity at different locations of the environment. The possible trajectories were:

- Moving in the meeting room (in a loop)
- Moving along the hallway (in a loop)
- Moving through the meeting room and exiting it through the kitchen in direction of location B (in the table abbreviated with M $\rightarrow$ K)
- Moving in the kitchen and then leaving through the hallway in direction of location A (in the table abbreviated with K $\rightarrow$ H)

The exact location of these places in the environment are given in figure A.7. The results of the simulation have also been validated with the real robot and real AmICA nodes, some of these runs with the real robot are presented at the end of this section.

For the following evaluation, a total of 400 experiment runs with different settings have been conducted. As already mentioned, a person was moving through the environment using different trajectories. This is the first parameter of the experiments. Then, the risk assessment has been either activated or deactivated. The results are shown in terms of numbers in table 5.3. The first column shown the location of the activity, the second column contains an $\mathcal{R}$ if risk assessment was active.

The following columns give statistics for all the runs of that class. For each experiment run, the mean of the distance d between robot and human and the mean of the cost value c during the traversal of the path has been recorded. For both the distance ($d_\star$) as well as the cost ($c_\star$), the table gives the minimum, mean, maximum and standard deviation ($\star \in \min, \mathrm{mean}, \max, \sigma$) over all the experiment runs of that class. For a complete view over all the experiment runs, refer to figure 5.18 which contains the single data points for the risk costs c for all eight experiment classes. In all runs, the robot came up with only two possible paths (which can be seen later in figures 5.19 and 5.20) – one through the meeting room and one through the hallway. These paths have almost identical length. The one through the meeting room constitutes the shortest path between the two positions.

The table shows several interesting aspects when, for one activity, the experiment runs without and with risk assessment are compared with each other. In case of activity in the meeting room, the mean value d_{mean} of the shortest distance between the robot and the human increases from 6.4m to 10m. The same aspect can be seen by looking at the mean cost c_{mean} of the runs. It *de*creases from 0.4 to 0.1. Besides the mean value, also the smallest value for the distance, d_{min} and the biggest value, d_{max} are larger when risk assessment is active.

Regarding the scenario with activity in the hallway, there is no observable difference in d_{mean} or c_{mean}. This is caused by the fact that the robot favors the slightly shorter way through the meeting room. Thus, even without risk assessment activated, the

Activity	**Flag**	**Runs**	d_{min}	c_{min}	d_{mean}	c_{mean}	d_{max}	c_{max}	d_σ	c_σ
Meeting	-	50	6.2m	0.36	6.4m	0.40	6.6m	0.43	0.06m	0.010
Meeting	$\mathcal{R}$	50	10m	0.10	10m	0.10	11m	0.11	0.03m	0.002
Hallway	-	50	9.1m	0.12	9.3m	0.13	9.4m	0.15	0.04m	0.006
Hallway	$\mathcal{R}$	50	9.1m	0.11	9.3m	0.12	9.4m	0.16	0.06m	0.008
M $\rightarrow$ K	-	50	4.1m	0.61	4.2m	0.64	4.5m	0.67	0.10m	0.014
M $\rightarrow$ K	$\mathcal{R}$	50	6.8m	0.31	7.0m	0.32	7.1m	0.35	0.06m	0.008
K $\rightarrow$ H	-	50	12m	0.00	13m	0.00	13m	0.02	0.34m	0.003
K $\rightarrow$ H	$\mathcal{R}$	50	10m	0.17	10m	0.20	11m	0.21	0.11m	0.005

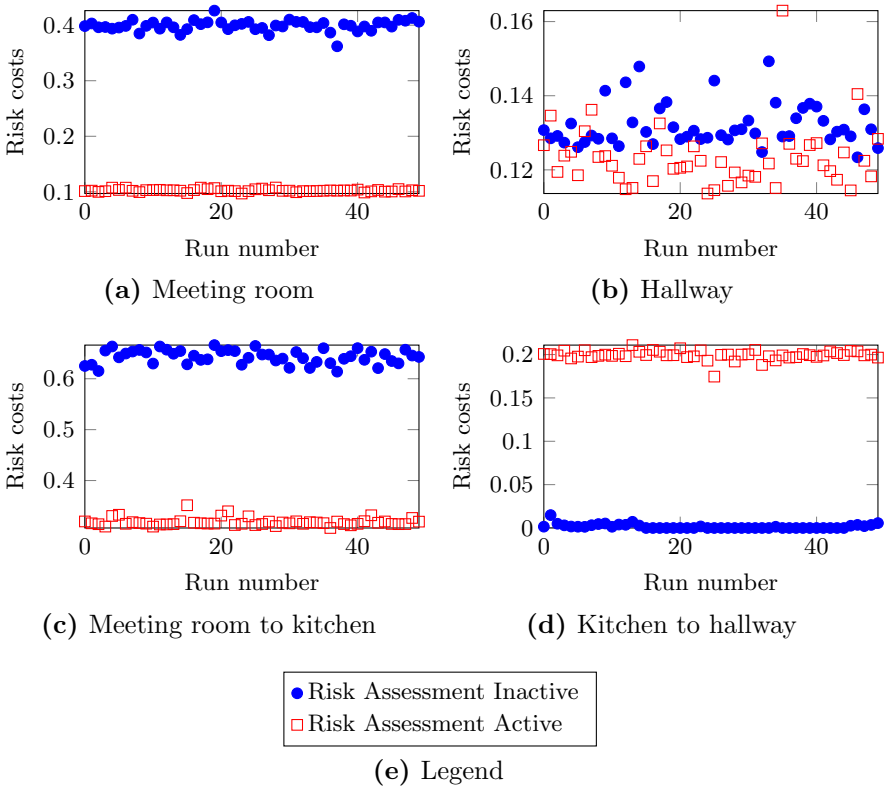
Table 5.3.: Analysis of several experiment runs with different activities and risk assessment activated and deactivated.

robot chooses the path through the meeting room. The activity in the hallway does not change that path in case of activated risk assessment.

Another example of a significant increase of distance (and thus safety) is the one where the person move from the meeting room through the kitchen (M $\rightarrow$ K). This example shows an increase of d_{mean} from 4.2m to 7m and a decrease of c_{mean} from 0.64 to 0.32. This is due to the fact that the robot chose paths through the hallway instead of through the meeting room in case risk assessment was active.

The last activity, from the kitchen to the hallway (K $\rightarrow$ H) reveals a downside of activated risk assessment. In this example, the mean distance d_{mean} is *de*creased from 13m to 10m and the mean costs of almost 0 are increased to a value of 0.2. While even those values still describe very low risk, compared to the other scenarios, it nevertheless gives an example in which activating the risk assessment decreases the safety of the system. This issue can be alleviated by enabling predictive path planning, as presented in the next section.

Additionally, the results of some exemplary experiments with the *real* robot are also given in a similar representation as in [Arndt 14]. Figure 5.19 shows an example run with activity in the meeting with risk assessment deactivated, figure 5.20 shows an example run with the same activity, but with activated risk assessment. For both figures, the real robot in the real aware environment has been used to estimate the state and plan the path. The initial situation (the belief at the time of planning) as well as the planned path is given on the left side of each figure. The right side shows an evaluation of the shortest distance between the robot and the human as well as the associated risk according to the risk function. This evaluation shows a dynamic view on the experiment as it contains several data points over the time, during which the robot has traversed the path. Distance as well as safety cost have been evaluated off-line. The ground truth of the human position has *not* been recorded during the



(a) Meeting room **(b)** Hallway

(c) Meeting room to kitchen **(d)** Kitchen to hallway

(e) Legend

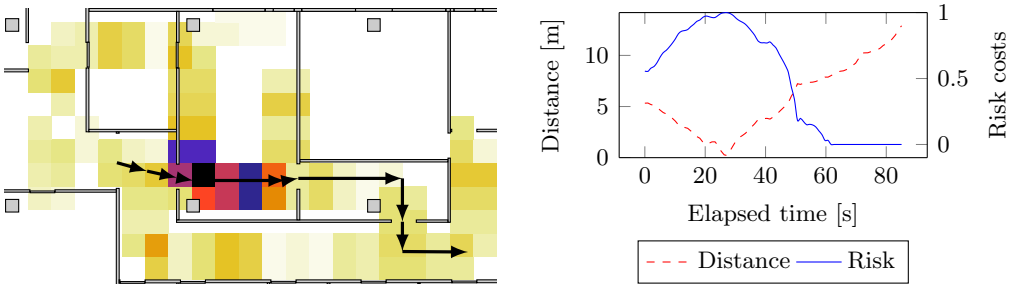
Figure 5.18.: Scatterplots visualizing the mean risk costs during each experiment run for all eight experiment classes. The axes are scaled so that they show all data points for each experiment class.

experiment. Instead, the position of the human has been assumed to be fixed (near the region of activity).

5.2.6. Discussion

This section has introduced the concept of risk-reduced path planning, as it is used in this thesis. Originating from a method to quantify risk (using a risk cost function), an algorithm to create a risk cost graph out of the currently estimated state of the environment has been introduced. This algorithms assigns high weights to edges that are close to (estimated) positions of people in the environment. By fusing this cost graph with a traditional distance graph, it is possible to use standard algorithms (e.g. Dijkstra's algorithm) to find safe and cost-efficient paths from start to goal positions.

Experiments conducted in simulation and reality have been used to evaluate the proposed methods. Depending on the scenario (i.e. the activity in the environment), this risk assessment can significantly decrease the risk during path traversal by staying



(a) Belief at the time of planning and planned path.

(b) Evaluation of distance and risk during traversal of the path.

Figure 5.19.: Example of a planned path and a traversal of the path *without* risk assessment. Activity is in the meeting room. The robot ignores this fact and traverses the meeting room nevertheless.

away from human beings. In the examples, the length of the paths is almost identical, thus the robot still operates efficiently. It must however also be noted that in one scenario (where the person is moving from the kitchen to the hallway), activating the risk assessment during path planning *increases* the risk of encountering a human. This is due to the dynamic movement of the person which the planner does not consider. This is handled in the next section, where *prediction* is added to the planning process.

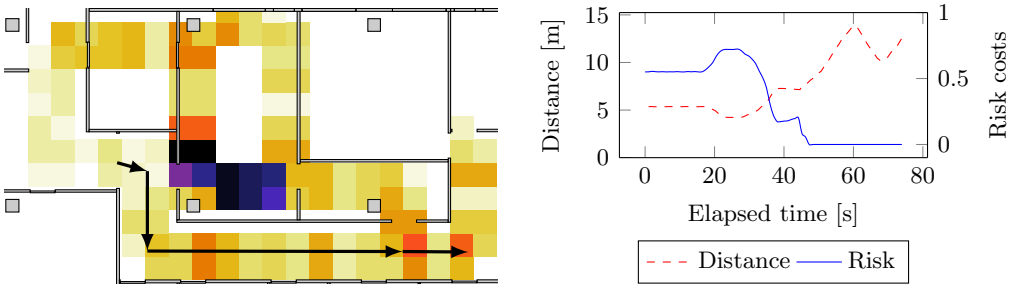
5.3. Predictive Path Planning

The idea of predictive path planning is an improvement over the work that has been introduced previously in section 5.2. The algorithm and experimental results have originally been published in the proceedings of the 12$^{\text{th}}$ International Conference on Informatics in Control, Automation and Robotics (ICINCO) 2015 [Arndt 15]. In this section, the approach and the results are replicated more thoroughly.

5.3.1. From Risk Assessment to Risk Prediction

In the previous section, it has been shown how to find optimal paths with respect to both safety as well as efficiency. The evaluations have shown that the method improves safety in general, but obviously, there exist some weaknesses by design.

To recall, it has been assumed that the environment stays static during the whole planning process *and* the execution of the plan. This assumption does usually not hold, because people move around and change their positions in most environments, especially in the office environment of the experiments. In this case, the paths planned using the previous method may not be optimal anymore with respect to the fused graph, which aims for low risk and efficient paths.



(a) Belief at the time of planning and planned path.

(b) Evaluation of distance and risk during traversal of the path.

Figure 5.20.: Example of a planned path and a traversal of the path *with* risk assessment. Activity is in the meeting room. The robot honors this fact and traverses the hallway instead.

This is the point where the concept of *predictive path planning* comes to the rescue. As the name already suggests, it enhances the previously introduced risk-reduced path planning with a predictor that aims to "look" into the future by trying to predict the future states during which the robot is traversing the path. Thanks to the way the tracking is realized (using probabilistic filters) this extension is surprisingly straightforward to realize, as can be seen soon.

5.3.2. State Prediction in Robotics

In mobile robotics, there exist many applications in which the prediction of futures states is of interest. For example, Foka and Trahanias [Foka 10] describe an approach tailored for mobile indoor robots that aims to simulate human behavior to improve obstacle avoidance. The authors acknowledge the ability of humans to predict the behavior of other people nearby, and aim to achieve similar capabilities for mobile robots. They explicitly predict and plan on two levels: A local short term prediction is used to predict only one small time step into the future. In contrast to that, the long term predictor aims to determine the goal point in the environment that a dynamic obstacle is targeting. The latter is the one that is comparable to the goals of the predictive planning in this thesis. This prediction is based on *hot points* in the environment that are known to be interesting to people, i.e. that are points where people might end their trajectories. These points can be either manually defined or automatically learned (off-line) by observing the environment.

An example with a very different application is located in the field of marine robotics. Alvarez et. al [Alvarez 04] have presented a genetic algorithm that can be used to find energy efficient paths for autonomous underwater vehicles. The ocean environments that have been regarded in that work exhibit time-varying currents. Their path planner operates in the spatio-temporal domain using forecasts of the ocean currents.

Using this spatio-temporal planning process, the authors are able to plan paths through underwater environments that are safe and require small amounts of energy.

5.3.3. Prediction using Bayesian Filters

To understand the state prediction, as implemented in this section, it is useful to shortly recap how the recursive Bayesian update step looks like. The update rule, as given in (4.3) on page 35, consists of two steps: First the prediction, which uses the motion model to predict the evolution of the state and second the correction, which incorporates current sensor measurements into the belief using the sensor model.

If the correction step is removed from the update rule, the new update rule (5.9), as initially published in [Arndt 15], arises. This update rule only uses the motion model in the prediction step and does not rely on sensor input data at all.

$$\text{belief}(x_t) = \eta \int p(x_t \,|\, x_{t-1}) \,\text{belief}(x_{t-1}) \,\mathrm{d}x_{t-1} \tag{5.9}$$

At this point, two important aspects should be noted. The first aspect is, that this prediction is *decoupled* from the online tracking. Whenever a new prediction has to be made, a copy of the current belief of the tracker is created and the prediction operates on this copy. This means that the prediction is *not* affecting the online tracking, no matter what approach is used for that task. Mentioning the tracking approach leads over to second aspect: Similarly as in the previous application (the risk-reduced path planning without prediction), it does not matter which state estimation algorithm is used to provide the input belief to the predicting approach. The only input that is needed is a belief that represents an estimation of the current state as good as possible – may it come from a single-target tracker, a probability hypothesis density (PHD) filter or a multiple hypothesis tracker (MHT). In either case, only the current belief (i.e. the current snapshot) is taken and the particles are moved according to the motion model. There is no interaction between targets.

5.3.4. Planning with Prediction

Now that a method for state prediction has been established, the path planning process for the robot itself is explained in this section. The path planning is based on the idea that a list of candidate paths from start to goal is evaluated for their safety (and length) and the one with least costs is selected to be traversed by the robot. The list of candidate paths can e.g. be generated by enumerating all the *simple paths* (simple paths have been previously introduced in section 5.1) from start to goal[14].

[14]The alert reader may be reminded of the previous section, where a very naive approach to find the safest path (without prediction) has been sketched, but discarded because of its lack of efficiency.

Each of these candidate paths $(v_0, \ldots, v_n)$ is analyzed using the procedure **Analyze-Path**, given in algorithm 5.4. The procedure is called with the path to be analyzed and the current belief at the time of planning, belief(x) as arguments. Further parameters to the algorithm include the step length for discretization l_{step} and the assumed velocity of the robot v_{robot}. Both are used together to calculate the time delta that presents the granularity of the prediction in line 3.

Algorithm 5.4: Function to analyze a whole path, algorithm originally published similarly in [Arndt 15].

Require: $l_{\text{step}} > 0$ {*step length for discretization*}
Require: $v_{\text{robot}} > 0$ {*assumed velocity of robot*}
Require: $(v_0, \ldots, v_n),\ n > 1$ {*a non-empty path to be analyzed*}
Require: belief(x) {*belief at time of planning*}
Returns: Total cost of the given path

```
1:  AnalyzePath((v_0, ..., v_n), belief(x)) :
2:  InitializeFilter(belief(x)) {initialize the local filter with the belief}
3:  t_step = l_step / v_robot
4:  {choose t and e to point to the beginning of the path}
5:  t ← 0
6:  e ← e_{v_0 v_1}
7:  c ← 0 {initialize costs}
8:  end_reached ← false
9:  while ¬end_reached do
10:     belief(x) ← AdvanceFilter(t_step) {update probabilistic filter for the simulated
        passed time}
11:     (end_reached, e, t, c) ← AnalyzeSegment(e, t, l_step, c)
12: end while
13: return c
```

What the algorithm does is (descriptively presented) the following: It simulates the robot traversing the path (driving forward) a small increment (l_{step}) and then advances the belief by predicting the state of the environment after that small time (t_{step}) has passed (see line 10). It then evaluates the cost of that driven segment, according to the current prediction. This is achieved by the procedure **AnalyzeSegment** in line 11. This process is repeated until the end of the path has been reached. The variable c (initialized in line 7) keeps track of the total costs of the path.

The procedure to analyze a single segment is called **AnalyzeSegment** and is given in algorithm 5.5. First, it calculates the new position t' on the edge (see line 3), which is done using the distance to travel d and the step length l. If the new position is still on the same edge ($t' \leq 1$), the process is simple, see line 18. In this case, only the fraction of cost, traversed on the edge must be summed up and the same edge with new parameter t' are returned. If however, the new position is beyond the

The content of this section is based on this naive approach. In fact, the predictive path planning turns out to be much less efficient (in terms of computation time) than the previous planner without planning.

Algorithm 5.5: Function to analyze a segment of a path, algorithm originally published similarly in [Arndt 15].

Require: $(v_0, \ldots, v_n)$ {*the whole path to be analyzed*}
Require: $d > 0$ {*the remaining distance to travel*}
Require: $\forall e_{ij} \in E : w_{e_{ij}} \geq 0$ {*all edge weights according to the current belief, fused with the distance weights*}
Require: **Successor**(e) {*function that returns the successor of edge e on the path being analyzed*}
AnalyzeSegment(e_{ij}, t, d, c) :
$l = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|$
$t' = t + (d/l)$ {*calculate new edge position*}
if $t' > 1$ **then**
  {*the calculated position is no longer contained in the current edge*}
  $c \leftarrow c + (1 - t) \cdot w_{e_{ij}}$ {*add the fraction of the weight traversed on current edge to the costs*}
  if $\neg\exists$ **Successor**(e_{ij}) **then**
    return $(\text{true}, e_{ij}, t, c)$ {*if there is no successor, the path has been traversed till the end*}
  end if
  $e' =$ **Successor**(e_{ij}) {*get the next edge of the path*}
  $t' = 0$
  $d' = d - (1 - t) \cdot l$ {*remaining length to travel*}
  if $d' < 0$ **then**
    return $(\text{false}, e', t', c)$ {*no more distance to travel, break out of recursion*}
  end if
  return **AnalyzeSegment**(e', t', d', c)
else
  $c \leftarrow c + (d/l) \cdot w_{e_{ij}}$ {*add the fraction of the weight traversed on current edge to the costs*}
  return $(\text{false}, e_{ij}, t', c)$ {*edge e_{ij} did not change, only the parameter t'*}
end if

current edge (in the case $t' > 1$), the process is slightly more complex. Additionally to summing up the fraction of cost on the current edge, the successor edge in the current path needs to be found and analyzed recursively in line 16, unless enough length has been traversed already or there is no successor (i.e. when the path ends).

Regarding the parameters, the robot velocity should be set to a typical value for the employed mobile robot. In the following experiments with the mobile robot Artos, it has been set to $v_{\text{robot}} = 0.5\text{m/s}$. The step length is less defined by the robot, but more by the layout of the environment (i.e. the graph used for tracking and path planning). Generally, a smaller graph (with shorter paths between start and goal positions) requires a smaller step length than a large graph with larger distance to travel. The size of the l_{step} also directly influences the run-time of the path planning algorithm. The smaller the step size, the more steps and thus the more predictions need to be done. For the experimental setup in this section, a step length of $l_{\text{step}} = 0.3\text{m}$ has been chosen.

5.3.5. Experiments

Also for the predictive path planning, several experiments have been conducted. As the path planning with prediction extends the previous, risk-reduced path planning, the experimental setup is identical to that of section 5.2, except that the planner uses its prediction capabilities.

The data in table 5.4 extends the results from the previous experiment, as given in table 5.3 by four more classes of experiments: the variants with activated prediction. Those classes are marked with a $\mathcal{P}$ in the column for the flag. It must be noted that for the runs that do *not* have prediction enabled, the same data as in table 5.3 is reproduced. This enables to easily compare the results with *enabled* prediction with the previous results. Similarly to the previously presented figure 5.18, the data from the single runs with enabled prediction is also presented in figure 5.21.

What can be seen from this table is a similar effect that was also visible in the analysis of the earlier experiment. Depending on where the activity was situated there is a smaller or larger effect on the mean values of the distance and the associated costs. With activity in the meeting room, the smallest distance between robot and human could be increased from 10m to 11m, when comparing the situation with only risk assessment with the situation with prediction. The scenario with the person moving from the meeting room to the kitchen (M $\rightarrow$ K) shows slight increases in the mean distance between robot and person, but the costs according to the cost function do also slightly increase.

However, the most interesting result is the change for the activity from the kitchen to the hallway (K $\rightarrow$ H), which did show a decrease of safety when risk assessment was activated in section 5.2. By taking the step from risk assessment to activated prediction, this can be "fixed". The mean distance d_{mean} can be increased from 10m to 12m. By looking at figure 5.21 it can be seen that most experiment runs have a

Activity	**Flag**	**Runs**	d_{min}	c_{min}	d_{mean}	c_{mean}	d_{max}	c_{max}	d_σ	c_σ
Meeting	-	50	6.2m	0.36	6.4m	0.40	6.6m	0.43	0.06m	0.010
Meeting	$\mathcal{R}$	50	10m	0.10	10m	0.10	11m	0.11	0.03m	0.002
Meeting	$\mathcal{P}$	50	10m	0.07	11m	0.09	11m	0.10	0.06m	0.006
Hallway	-	50	9.1m	0.12	9.3m	0.13	9.4m	0.15	0.04m	0.006
Hallway	$\mathcal{R}$	50	9.1m	0.11	9.3m	0.12	9.4m	0.16	0.06m	0.008
Hallway	$\mathcal{P}$	47	9.1m	0.13	9.2m	0.14	9.3m	0.17	0.07m	0.009
M $\rightarrow$ K	-	50	4.1m	0.61	4.2m	0.64	4.5m	0.67	0.10m	0.014
M $\rightarrow$ K	$\mathcal{R}$	50	6.8m	0.31	7.0m	0.32	7.1m	0.35	0.06m	0.008
M $\rightarrow$ K	$\mathcal{P}$	50	7.1m	0.32	7.3m	0.34	7.7m	0.35	0.11m	0.005
K $\rightarrow$ H	-	50	12m	0.00	13m	0.00	13m	0.02	0.34m	0.003
K $\rightarrow$ H	$\mathcal{R}$	50	10m	0.17	10m	0.20	11m	0.21	0.11m	0.005
K $\rightarrow$ H	$\mathcal{P}$	50	10m	0.00	12m	0.03	13m	0.22	0.98m	0.078

Table 5.4.: Analysis of several experiment runs with different activities, activated risk assessment and additionally activated prediction.

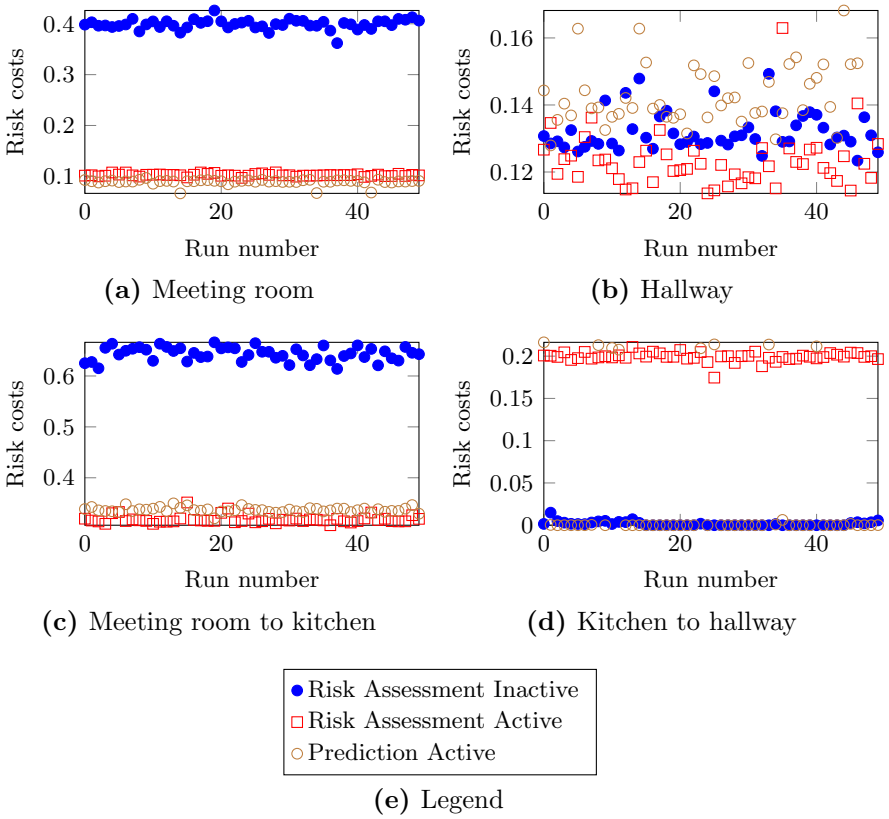
risk cost of approximately zero while a few have costs similar to the previous situation with only risk assessment activated. For all of these eight runs, the robot has chosen a path through the hallway, just as it has always done if only risk assessment was active. The other runs led the robot through the meeting room as expected.

For a comparison of the predicted situation and the situation in *reality* (using the real robot), see figure 5.22[15] The upper row contains the predictions, the lower row the situation in reality. While, according to the prediction, the belief may be broadly distributed at the end of the path traversal, in reality, the person stays in the meeting room and the belief is thus still concentrated there.

5.3.6. Discussion

Motivated by the observation that the environment is usually dynamic during the traversal of the path by the robot in section 5.2, this section has extended the previous idea by a *prediction* of the evolution of the environment. Using the already existing motion model (as defined in section 4.4), algorithms that estimate the future state

[15]Note that this comparison does not show exactly the same *times* of the situations in prediction in reality. Instead, it compares the situations in which the robot positions roughly match. This method is necessary as traversal of the path in reality does obviously not perfectly match the predicted traversal.



(a) Meeting room **(b)** Hallway

(c) Meeting room to kitchen **(d)** Kitchen to hallway

(e) Legend

Figure 5.21.: Scatterplots visualizing the mean risk costs during each experiment run for all twelve experiment classes. The axes are scaled so that they show all data points for each experiment class.

of the environment while the robot is hypothetically traversing its path have been presented.

The experiments (again in simulation and reality) have shown little improvements in situations in which the risk-reduced path planning did already perform well. However, in the situation in which the risk-reduced path planning (without prediction) *did* fail, the *predictive* path planning did indeed plan safer paths.

5.4. Robot Localization in Aware Environments

The application of using aware environments to assist mobile robot localization differs from the other applications introduced so far, as it does *not* make use of the framework for estimating the positions of people which was established in chapter 4. Still, it is relevant for the goals of this thesis, as can be seen shortly. Some of the ideas in

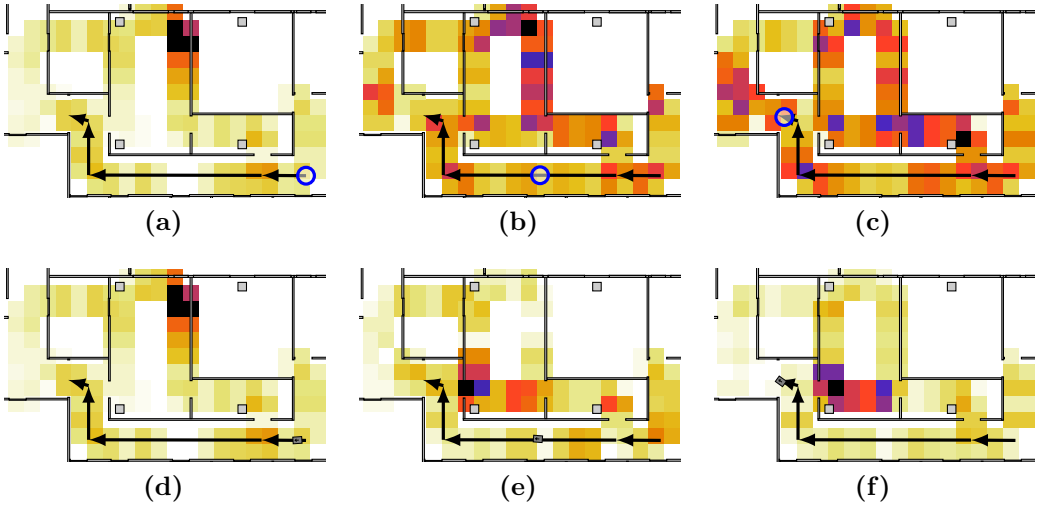


Figure 5.22.: Comparison of predicted traversal of the path (with the predicted evolution of the state) and path traversal in reality. The upper three images shows the predicted situations for the robot at the positions indicated by the circles. The lower images show the situations in reality.

this section data back to an internal report [Arndt 12a] and a publication at the 22nd Conference on Autonomous Mobile Systems (AMS) 2012 [Arndt 12b].

5.4.1. Relation to this Thesis

In the chapter about fundamentals, different methods for mobile robot localization have been introduced (see section 2.1.2). However, many popular (and inexpensive) indoor localization methods for mobile robots, e.g. the Monte Carlo localization introduced in section 2.1.2.3 do not provide *global* localization, i.e. they do not guarantee to provide exact localization estimates if the robot is switched on and not provided with any initial localization information[16].

At exactly this point, the localization using radio signals from transmitters in the aware environment can provide a (potentially very rough and inaccurate) initial estimation of the robot's position. This rough estimate can then later be refined by incorporating distance-sensor measurements by the Monte Carlo localization. It is of high relevance to the ideas of this thesis, because it supports the idea of improving the robot's performance with technology that is available anyway without introducing additional installation or maintenance costs.

Of course, wireless transmitters are no essential ingredients of aware environments, but in many cases, there is wireless technology in use (like the AmICA nodes used

[16]The wake-up robot problem.

in this thesis). Strictly speaking, the methods introduced in this section do not even require the environment to be *aware* or *smart* according to the definitions in section 2.2. Instead it really only relies upon being able to receive and identify wireless transmitters whose positions are known (and which are usually static). Those could e.g. also be WiFi access points or similar technology.

5.4.2. Fundamentals of Radio Communication

The task of estimating the position of a mobile radio receiver within a network of (fixed) transmitters is not new at all and has been researched for a multitude of applications. The methods can be split into two major categories. The localization using *triangulation* and the ones using *Received Signal Strength (RSS)* fingerprinting. However, before advancing further, some fundamentals regarding radio communication need to be reviewed, as they form the foundations of both of these categories.

The maximum range of radio communication links is limited, as a transmitter can only transmit with finite power. This power is in most cases also limited by regulations and laws. When radio waves propagate, the power that can be received decreases with the distance to the sender. A receiver can also not be built with unlimited sensitivity, so there is an obvious limit to the communication range.

The so called *Free-Space Path Loss* model (see e.g. [Blaunstein 07, p. 97], [Boccuzzi 08, p. 150], [Parsons 92, 2.2]) is a well-known model to describe these losses over distance in a free space (i.e. a space not obstructed by any obstacles and without anything influencing the waves' propagation). In this work, the notation given in (5.10) is used to determine the loss L using the wavelength λ and distance d between transmitter and receiver as input. This can be rewritten as in (5.11) for the frequency f as an input instead of the wavelength λ due to the relationship $\lambda = v/f$ with v being set to the speed of light c.

$$L = \left(\frac{4\pi d}{\lambda}\right)^2 \tag{5.10}$$

$$= \left(\frac{4\pi df}{c}\right)^2 \tag{5.11}$$

In typical radio applications, the frequency f can be approximated to be fixed (it only varies by the bandwidth of the signal, which is however typically small compared to the center frequency f). The frequency along with the constants 4π and the speed of light c can then be combined into a single constant, so that the loss can be expressed as a function proportional to the squared distance d^2 between transmitter and receiver, see (5.12).

$$L = \left(\frac{4\pi f}{c}\right)^2 d^2 \tag{5.12}$$

A different source of losses is diffraction. Diffraction happens when a (radio) wave hits the edge of an obstacle [Parsons 92, 3.3] and the radio waves are "bent" around that edge, see figure 5.23 for a visualization. Diffraction directly leads to loss which is called diffraction loss and is dependent on the position and size of the obstacle, but it also leads to further radio phenomenons, such as multipath propagation [Parsons 92, 5.1] by altering the wave.

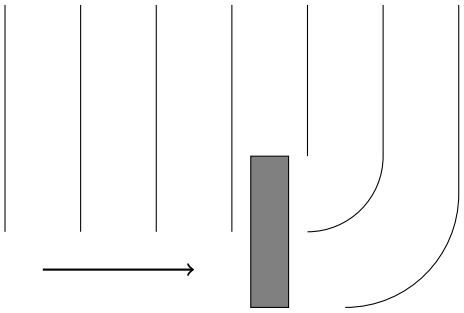


Figure 5.23.: Diffraction of radio waves at an obstacle. Drawing inspired by [Parsons 92, Fig. 3.2].

Another important behavior of radio waves is multipath propagation. When there is not a single, unique path that the radio waves take between transmitter and receiver, this is called multipath propagation. Causes for multipath propagation are the already mentioned diffraction and also the reflection of radio waves on obstacles. All these phenomenons split up the waves and alter the distances they have to travel till they reach the receiver. Depending on the phase shift between the signals, this may cause either constructive or destructive addition of the waves at the receiver [Parsons 92, 5.2], potentially making reception impossible. In figure 5.24, such an example situation is depicted.

While in principle, (5.10) gives a formula to estimate distances between transmitter and receiver using a measure of the received energy, this does not work well if the "free space" assumption is violated and the effects that have previously been named become relevant. Even Parsons acknowledges that "None of the simple equations [...] are suitable in unmodified form for predicting average signal strength in the mobile radio context [...]" [Parsons 92, 3.1].

Especially in indoor environments, the "free space" assumption is heavily violated for most frequencies, because there are floors, ceilings, walls and many other obstacles that influence the radio waves and make predicting their propagation very hard (as can be seen in the example later). For this reason, localization by triangulating a number of

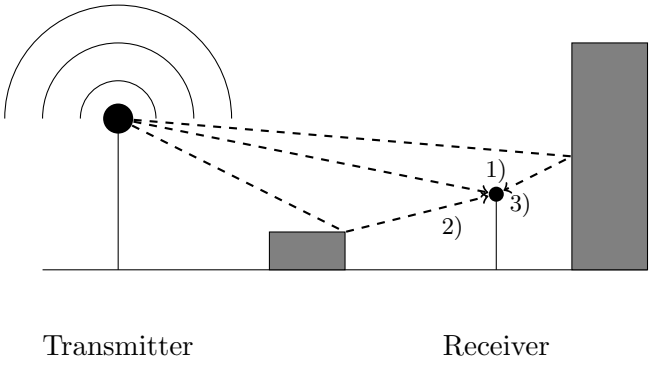


Figure 5.24.: Visualization of possible paths 1), 2) and 3) during multipath propagation of radio waves.

transmitters whose positions are known, is a very naive and a non-promising attempt. The author has tried the triangulation with AmICA nodes in indoor environments. It did perform poorly [Arndt 12b, 3.4].

Fortunately, the method of *Received Signal Strength (RSS)* fingerprinting (or RSS profiling as it is also sometimes called [Mao 07, 2.3]) comes to the rescue. It also uses the indicators of the energy received from the transmitters, but instead of trying to map these values to absolute distances, it uses "fingerprints" of received signal strengths from different transmitters to infer (previously learned) positions in the environment.

When using RSS fingerprinting methods, the positions of the transmitters do *not* need to be known. Instead, for each position where localization should be possible, there needs to be a known fingerprint of the transmitters in range. When localization starts at an unknown position, the receiver first has to build up its own fingerprint for that position by sampling a certain number of frames or for a certain period of time. This fingerprint is then compared to the ones contained in a database and the best matching location can be returned as the result of the localization process[17].

There are numerous applications of RSS fingerprinting in academia (see e.g. [Shin 10], [Honkavirta 09] or [Widyawan 07] to list just a few) as well as in commercially available solutions. The latter are relevant for localization of mobile devices (such as mobile phones, tablets or similar computing devices) in the absence of, or while assisting global navigation satellite localization systems (see section 2.1.2.1 about such systems). At the time of writing, mobile phones based on Apple Inc.'s iOS, Google Inc.'s Android and the browser Firefox by Mozilla Foundation are a few examples for the use of RSS fingerprinting with 802.11 WiFi access points and possibly other sources.

[17] Depending on the algorithm, there may be several candidates, and depending on the application, these other candidates may also be interesting, as shown later.

5.4.3. Learning the Radio Map

The database that contains fingerprints for all locations is also called *radio map*. It is a data structure that contains previously learned fingerprints for positions in the position space (this could be of Cartesian nature, but other coordinates may also be possible).

In this work, the radio map is implemented using a grid map structure with a two-dimensional Cartesian state space. While this has the advantage of being a straight forward implementation that relies on existing and well-tested code, it is not an appropriate solution for very large state spaces, as the (unoptimized) lookup of fingerprints requires a linear search through the whole map. Each cell of this grid map contains another data structure that holds a mapping from transmitter identifier to a tuple which contains a) the number of samples received from that transmitter and b) the arithmetic mean of the RSS value of that transmitter. In case of use with the AmICA framework, the identifier is equal to the source of the transmitter (which is an 8 bit integer) and the RSS is represented by the RSSI value that was measured at the receiver during reception of that frame. See figure 5.25 for an illustration of the structure of the radio map.

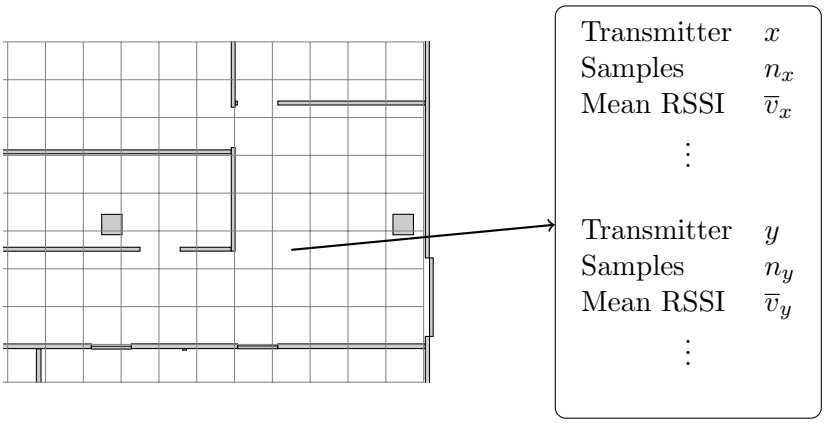


Figure 5.25.: Illustration of the radio map structure used in the thesis.

Whenever a valid frame is received during the learning phase of the localization system, there are two possible situations: If for this cell (at the current receiving position in the map) no frame from the transmitter t has been received yet, the sample counter is set to one ($n_t = 1$) and the mean is set to the RSSI value v of the received frame: $\overline{v}_t = v$. In the other case, the value of the mean is updated using (5.13), with $\overline{v}_n$ being the previously calculated mean after n samples and v_{n+1} being the currently measured RSSI. Afterwards, the counter is increased by one ($n \leftarrow n + 1$). This formula can be easily derived, see section B.6.

$$\overline{v}_{n+1} = \frac{(\overline{v}_n \cdot n) + v_{n+1}}{n+1} \tag{5.13}$$

As already mentioned in the introduction, the purpose of the radio signal localization system in this thesis is *not* to give precise localization results, but only course, global estimates, that can then be used to initialize the belief of a Monte Carlo Localization system. As soon as the Monte Carlo localization system has converged to a stable robot position (using sensor measurements and possibly also using the input from a dead reckoning system), it can be assumed that a (rather precise) location of the robot is known.

This fact can be exploited in the *learning phase* of the radio signal localization to automate the learning process. Instead of recording radio frames and manually labeling them with the (manually measured) ground truth, frames can automatically be recorded, even while the robot is moving, and be labeled using the (rather precise) pose as estimated by the Monte Carlo localization. This eases the process of radio map generation and even allows to do online adaptation of the map while the robot is already fully operational.

5.4.4. Localization within the Radio Map

The localization within the radio map works as follows: First, the robot waits till a configured number of frames[18] has been received at the current position. During the reception, a sample which uses the same data structure as one cell of the radio map is created. At the end of sampling, it contains mean RSSI values for all received transmitters. Then, for each cell of the radio map, the distance between the contents of the cell and the sample is calculated. Using this method, for each cell in the radio map a distance value is generated.[19] Depending on the application, only the coordinate of the cell with the least distance can be returned to the user (deterministic case) or the coordinates of the best n cells can be returned, preferably sorted by the distance.

To determine the distance between a cell and the sample (which has the same structure as a cell, so basically, to determine the distance between two cells), a metric is needed. In this work, the metric is defined by algorithm 5.6. The value of the penalty constant is dependent on the magnitude of the RSSI values. In this work, it has been chosen as $p = 50$.

[18] In the experiments a number of 20 was chosen for this parameter.

[19] This of course only works for *small* radio maps. If very large radio maps are used that span e.g. a whole building or several buildings, then this search for the minimum distance needs to be optimized.

Algorithm 5.6: Metric to determine the distance between two cells of the radio map.

Input: Two cells, C_1 and C_2
Output: Distance between the two cells

$d \leftarrow 0$ {*Start with zero distance*}
for $(n_x^1, \overline{v}_x^1) \in C_1$ **do**
  if $(n_x^2, \overline{v}_x^2) \in C_2$ **then**
    {*A mean RSSI for sender x is also available in C_2. Add the absolute value of the difference between the two to the distance.*}
    $d \leftarrow d + |\overline{v}_x^1 - \overline{v}_x^2|$
  else
    {*There is no value for sender x available in C_2, punish this fact by adding a constant penalty p to the distance.*}
    $d \leftarrow d + p$
  end if
end for
return d

5.4.5. Experiment

This section shows an example experiment related to the localization in an aware environment using RSSI values. The experimental setup is identical to the one used in the previous experiments – the (real) mobile robot Artos (see section A.2) together with AmICA nodes placed at the positions given in section A.3.

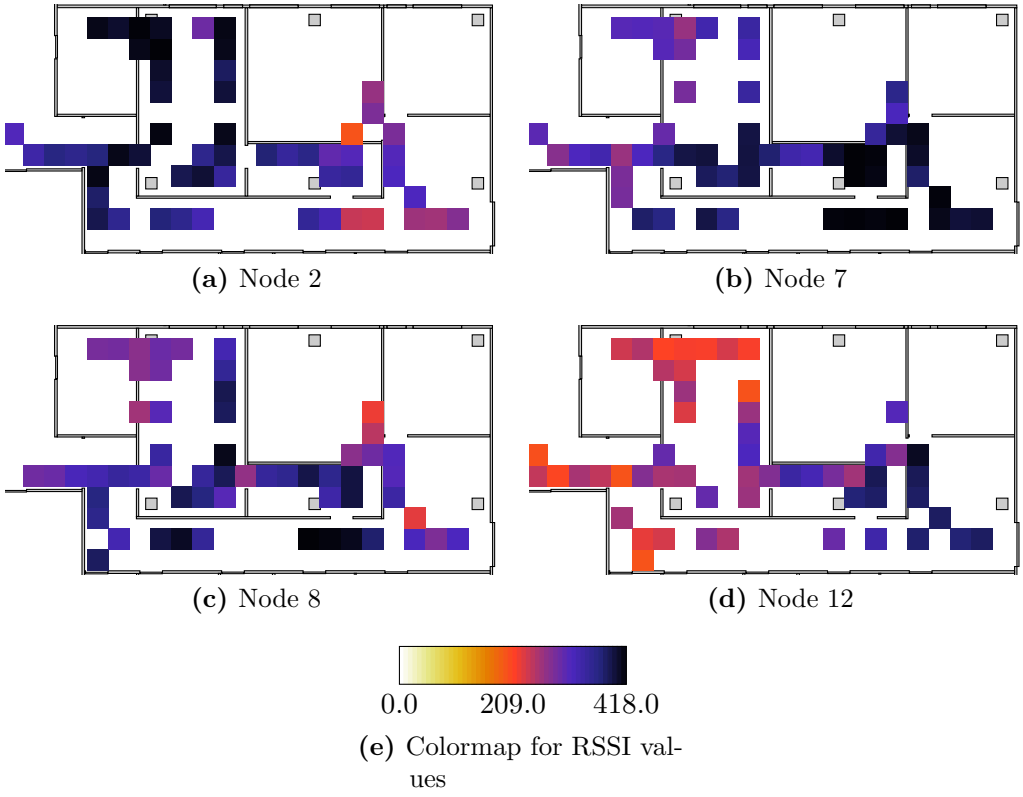
Training data has been collected for roughly 15 minutes. During that time the robot has been manually directed through the environment where it was stopped for some time at several positions. The whole time the robot was recording the frames, together with the position where they have been recorded. This localization was realized using the Monte Carlo localization algorithm running on the robot (see section A.2.5 for details). The localization process was manually supervised to ensure the correctness of the position estimations. During that time, 1945 frames from 9 distinct sensor nodes have been received[20]. For the exact distribution of the received frames, please refer to table 5.5.

Due to the high dimensionality, the complete radio map learned from these frames (as described earlier) cannot be easily visualized. For a single sending node, the radio map can however be visualized by showing the mean RSSI values for that node in a map structure. Four single maps of received signal strengths for the named nodes are exemplarily shown in figure 5.26. By carefully examining these maps (and maybe trying to infer the sensor mounting positions), the reader may easily see that the distribution of the received signal strength and the propagation of the radio signals is indeed hard to predict.

[20]Sensor node 1 did unfortunately fail during the experiment.

Sensor	Frames	Sensor	Frames
1	-	9	184
2	202	10	63
6	256	11	416
7	233	12	183
8	197	255	211

Table 5.5.: Distribution of received frames in the training set for RSSI localization.



(a) Node 2 **(b)** Node 7 **(c)** Node 8 **(d)** Node 12 **(e)** Colormap for RSSI values

Figure 5.26.: Visualization of radio maps for single senders. For placements of the senders in the environment, please refer to figure A.10.

After the collection of these training frames, the robot was again manually directed to some positions in the environment where frames have been collected to later evaluate the localization algorithm. Some example results for classifications can be found in figure 5.27. In that figure, the colors do no longer represent RSSI values, but the distances of the fingerprint of the cells to the fingerprint of the sampled frames for that position during the classification run. The darker the color, the higher the

distance and thus the lower the probability of the receiver/robot being situated at that position. The ground truth (the actual receiver/robot position) is indicated by the robot icon. The result of the classification has been sorted in ascending order by the distance. The numbers in the circles on cells represent that order. Only the ten best estimates have been marked.

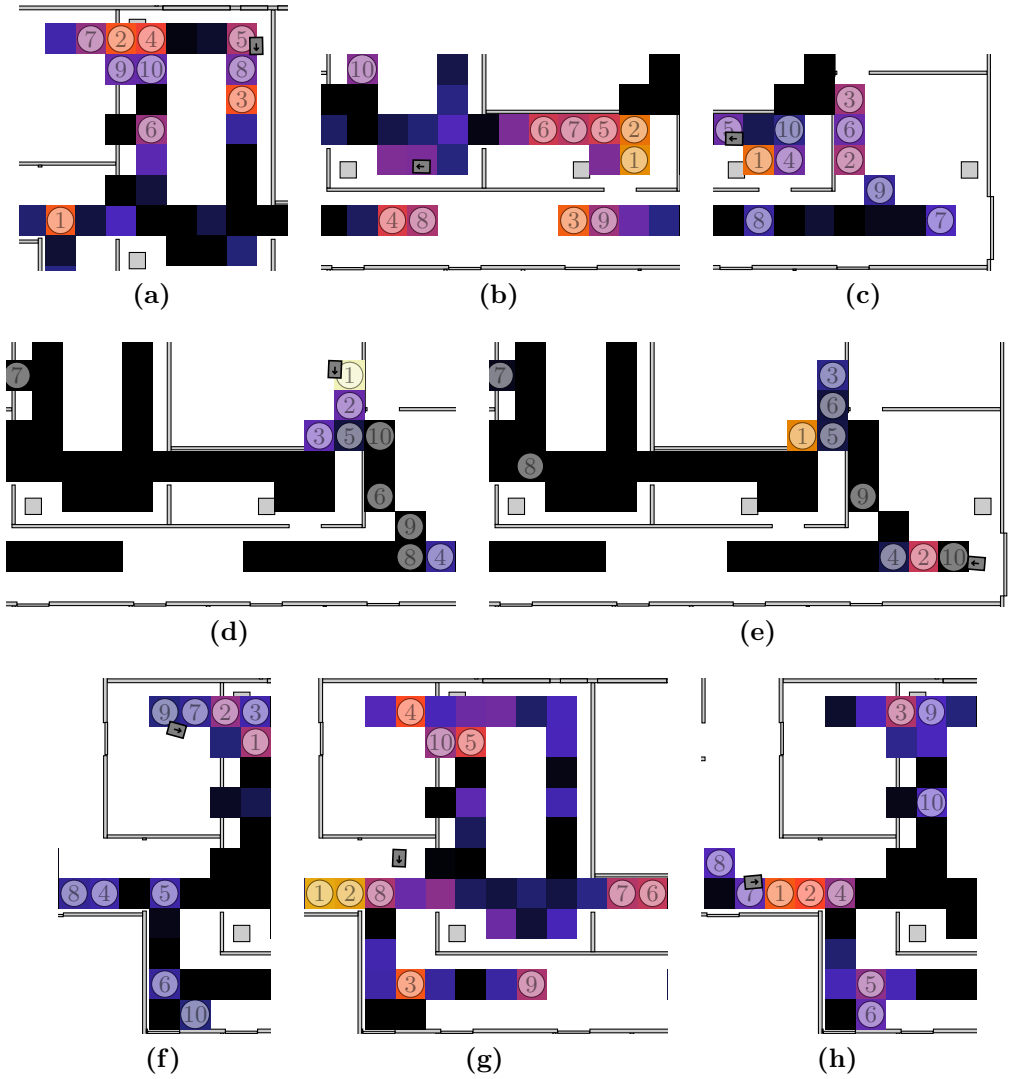


Figure 5.27.: Example localizations at different positions in the environment. The robot icon indicates the ground truth, the numbers in circles the order of the estimates, starting with the best. Although the figures have been cropped for brevity, care has been taken that the ten best estimates are still always shown.

By examining the figure, it can be concluded, that sometimes, the *best* localization results (as indicated by the number 1) are quite good, but sometimes, they are

relatively far off. In case the first estimation is far off, the next estimations are usually very close to the true position. It should be noted again that the result is used for the coarse initialization of a Monte Carlo localization algorithm. Thus no *definite* (i.e. deterministic) answer to the localization problem is needed. An example of the initialization of the Monte Carlo localization algorithm with this data is not given in this section, but in the later application study (see section 6.2.1).

5.5. A Second Robot

Back in section 4.5.9, the idea of extending the state estimation system with inputs from other sensors has been discussed. In this section, a second mobile robot is added to the aware environment. Both robots are able to communicate over a standard IEEE 802.11 WiFi wireless radio link and can exchange external (e.g. camera images or distance scans) as well as internal (e.g. the localized position estimation of the robot) sensor data. The goal is to be able to improve safe and cost efficient navigation in a scenario, in which there are multiple robots present anyways, with only minimal additional costs, by incorporating information from other robots into the overall system.

5.5.1. Knowledge Exchange

When two or more robotic agents are connected through a network, it must be defined what data is exchanged between them. While it is in principle possible to exchange raw and unprocessed or just slightly filtered sensor data, this may cause a high load in the network as well as on the receiving robot, which needs to process the data locally (this effect is worse, the more robots are present in the scenario). Instead, it is more efficient to only consider the data that is of actual interest in the intended application scenario and on which (possibly expensive) processing has already been performed.

In the context of safe navigation, the interesting aspect is to know where a human (or another dynamic obstacle) is currently present. This is basically what also the wireless sensor network nodes provide, but each robot may have much richer and more precise data available than the simple nodes. For example, the robots have completely different means for detecting people. Algorithms can be used to detect people's poses and even further, their velocity vectors out of camera images or range scans. Together with the estimated pose of the robot itself, the absolute pose of the human can be calculated out of the pose relative to the sensors.

There are many, partly highly sophisticated, methods available and described in the literature that are able to detect human beings using standard sensors mounted on mobile robots. There exist works describing how to detect humans in still images or video streams [Dalal 06], thermal infrared cameras [Fernández-Caballero 10] and also using RGB-D data [Spinello 11], to mention just a few.

Instead of implementing one of these complex human detectors described in the literature, the proof of concept in this work employs a much more simple idea. The mobile robot ARTOS that is used as a sensor (see section A.2 for a detailed description of the robot) processes its local sensor measurements (in this case from a RGB-D camera and a laser rangefinder) and uses a robot-centric occupancy grid map with polar coordinates as an intermediate representation of the values. This occupancy grid map detects not only dynamic, but also static obstacles, which are mainly the walls of the environment. As however, on the receiving robot, the state estimation is done exclusively on a graph, and the edges of the graph represent statically free space, the classification of walls as "obstacles" does not interfere with the state estimation.

The exchanged messages between the robots then consists of a tuple containing the current robot pose in global world coordinates and the local, robot-centric map M which is represented as a set of cells $m_i \in M$:

$$((x, y, z, \alpha, \beta, \gamma)^T, M) \tag{5.14}$$

The receiving robot can then decode these messages to know the poses of other robots and their local maps. These maps can be incorporated into the receiving robot's representation of the world. Additionally, the other robots' poses *themselves* may provide meaningful data, namely the positions of other robots which do constitute (possibly dangerous) dynamic obstacles.

5.5.2. Modifications to the Sensor Model

In case of the probabilistic state estimation, as proposed in chapter 4, once these tuples are received on the second robot, it must incorporate these measurements into its own belief about the state of the environment. Thus, the second robot as a sensor must also be modeled in the sensor model used for the probabilistic state estimation (see also section 4.5). While the sensor model for the passive infrared sensors discussed earlier are of a static nature (the sensor positions are not expected to change, once deployed), the second robot constitutes a dynamic sensor. This dynamic property has to be handled in the sensor model.

For the process of creating an occupancy graph (as described in section 4.8), the received data is incorporated as follows: For each particle on the occupancy graph representing part of the belief, the particle's position is first transformed into the coordinate system of the *other* robot's local map (as specified in the tuple). The transformed position is then checked for inclusion in the local map and if it is contained, the particle itself is updated with the probability for occupancy of that position in the map. See figure 5.28 for a visualization.

This figure shows the particles of the occupancy graph with their diameter representing their weight. It can be clearly seen that cells of the external occupancy grid map

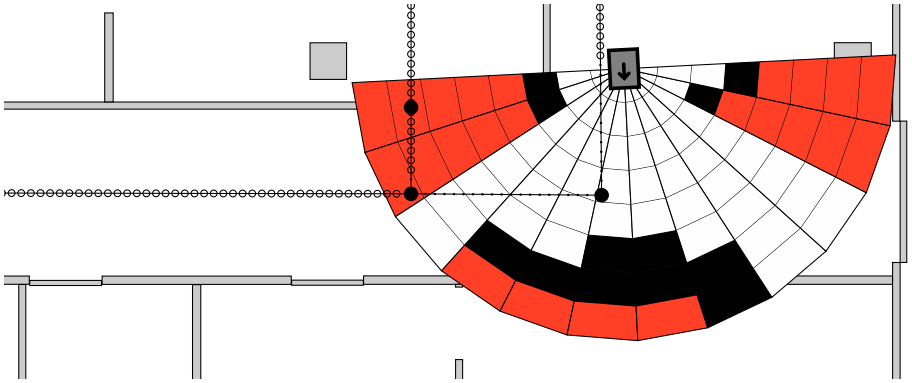


Figure 5.28.: Example of incorporating the local occupancy grid map of another robot into the occupancy graph. The image is generated from the local state of the *first* mobile robot ARTOS during an experiment. The position of the first robot is not visible within the map. The robot icon in this figure marks the position of the *second* mobile robot.

with a low probability of being occupied (the white cells) modify the weights of the particles to a very small value (they are barely visible). The cells with high probability (the black cells) match the walls of the environment and other obstacles (to the left of the robot). These cells would increase the weight of particles, but in this example these cells do not cover the graph and thus no particles. Cells drawn in red have a weight of approximately 0.5, they represent "unknown" states and do not modify the particle weights.

This process can be generalized to include information from other robots completely into the sensor model by taking the original model for the passive infrared sensors and extending it with support to include the dynamic occupancy grids at dynamic sensor poses. The modified sensor model can be visualized like in figure 5.29, where additionally to the statically mounted nodes, the dynamic pose of another robot and the accuracy of its local occupancy grid map is included.

5.5.3. Experiment

As the second robot is modeled similarly as other ambient sensors, only a single experiment has been executed to show the general feasibility of the approach and the extensibility of the system as a whole. This experiment is of clearly qualitative nature as it does not give any numbers as results. No further statistical analysis based on the experiment has been planned or conducted.

For simplicity, the robot that acts as the sensor that provides inputs to the other robot, has been used as a still-standing "observer", i.e. it was not moving during an experiment run. For multiple runs, this robot was placed at different, fixed positions in the environment. The other robot, similarly as in the previous experiments was

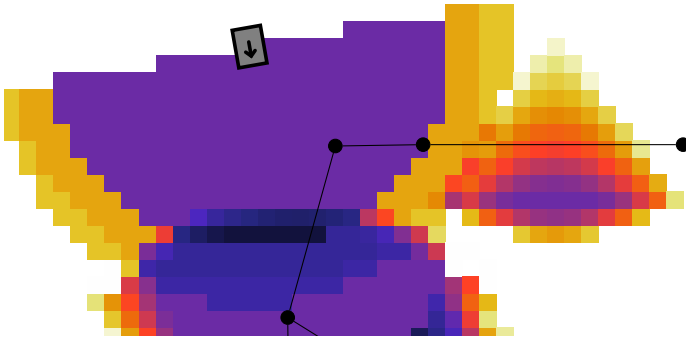


Figure 5.29.: Visualization of an overall sensor model including information from a second robot.

given a transportation task between two positions in the environment. To demonstrate the failure safety of the concept, during one run, a network failure has been simulated by disabling the network connection of the second robot.

As already stated, no statistics have been created about how well either the tracking or the virtual obstacle avoidance using the occupancy graph perform with the additional data. Instead, exemplarily, one experiment run (the one with the simulated network failure) is documented using figure 5.30. The local occupancy grid map of the second robot is visualized similarly as in figure 5.28.

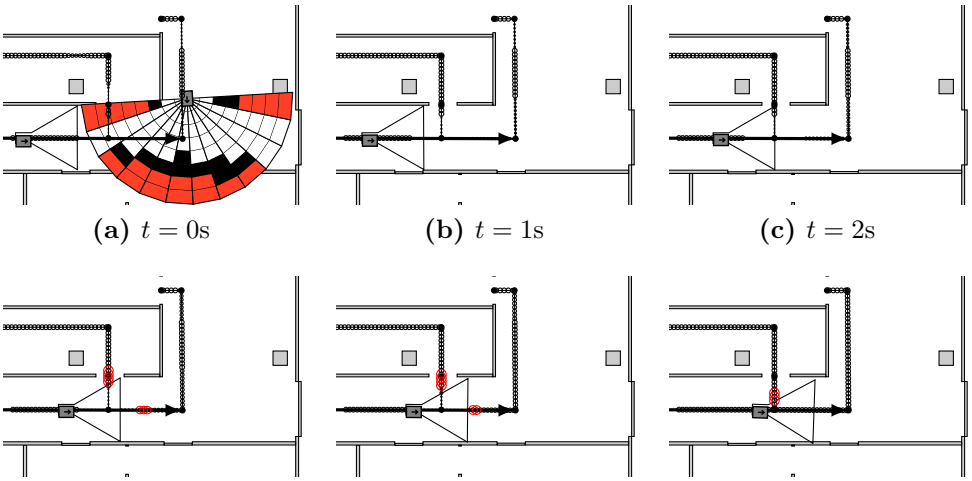


Figure 5.30.: Visualization of one experiment run with a second robot as input to the state estimation.

In the first subfigure, the connection between both robots has already been interrupted

(this could e.g. be the case if the network or the whole second robot fails or is switched off), but the occupancy grid map of the second robot is still valid (there is a small timeout to compensate for network latency). It can be seen that the particles that are observed by the second robot are still low-weighted. In the following subfigures (captured at the times given relative to the first one), the timeout has occurred and the map is no longer shown to visualize this fact. More importantly than this visualization of the failure, the occupancy grid map of the second robot is no longer used for state estimation by the first robot. It can be seen that the weights of the particles which are no longer covered by the second robot's map gradually increase to show that these regions are no longer classified as "free". At that point of time, the first robot has to rely on the other remaining sources for state estimation (i.e. the wireless sensor nodes).

The experiment shows that the second robot can indeed serve as a dynamic, mobile sensor that provides information to the first robot. At times with bad or missing network connection, the robot falls back to the situation in which only the input from the AmICA nodes is available. Although it has not been explicitly evaluated, it should be clear that the additional information can be used to increase the quality of the state estimation process. In principle it would also be possible to let the second robot move, as the detection of (dynamic) obstacles does not rely on the assumption of a standing robot.

6. Application Study

The previous two chapters have shown methods that have been used or developed to achieve the goals of establishing safe and cost efficient navigation in aware environments (cf. section 3.3). Additionally, they have presented previously published as well as new results. In this chapter, all the single aspects are brought together by regarding them in one large experiment.

6.1. Experimental Setup

This final experiment puts together all the single aspects that have been previously discussed and analyzed separately. The experimental setup is therefore very similar to the previous ones, except that in the application study, only the real system and no simulation has been used. As an aware environment, the offices of the Robotics Research Lab, outfitted with AmICA wireless sensor nodes are used once again. For the layout of the environment, and the placement of the sensor nodes, please refer to figure A.7 and figure A.10 in section A.3. The graph used for state estimation as well as for path planning is also contained in that section, see figure A.9.

The storyline of the application study is the following: The secondary robot plays a passive role in the experiment, as it is not actively moving through the environment. However, its task is to assist the primary robot, which has to actively move through the environment to fulfill its purpose. To achieve this, the secondary robot is placed at a fixed position in the corner of the south-western part of the hallway, not far from place A. That second robot is switched on, so that its local sensors are able to observe part of the environment which is only poorly covered by AmICA nodes. The expectation is that performance and safety can be increased for this region by the passive robot, as sketched in section 5.5.

At the beginning of the experiment, the user brings the primary robot to an initial position in the environment (which is not known to the robot) and switches it on. At this point, the robot uses the method for global localization using received signal strength indicators of the AmICA nodes (cf. section 5.4) to initialize its Monte Carlo localization algorithm using the coarse information about its whereabouts.

As soon as the localization has stabilized, the robot uses its implementations of the state estimation techniques introduced in chapter 4 to improve safety as well as performance during further operation. It uses the data with the virtual obstacle avoidance sensor to overcome the low velocity limits imposed by its own, short-ranged

local sensor systems (cf. section 5.1) and also to optimize the path planning with respect to a trade-off between performance and safety (cf. section 5.2 and section 5.3).

For the rest of the experiment, the robot is given several navigation tasks (similar as in the earlier experiments, between the points A and B). During all that time, one person is present in the environment and behaving according to some typical actions to generate activity. The externally visible state as well as parts of the internal state of the mobile robot are recorded during the experiment to evaluate its behavior off-line at a later time.

6.2. Experimental Results

The results of the application study have not been evaluated quantitatively, e.g. in form of a measure of the safety of chosen paths or similar. This has been done extensively in the previous chapters using experiments in the simulation as well as in the real environment. Instead, the results of the application study are presented in a more illustrative and qualitative way, in chronologic order according to the storyline that has been outlined in the previous section.

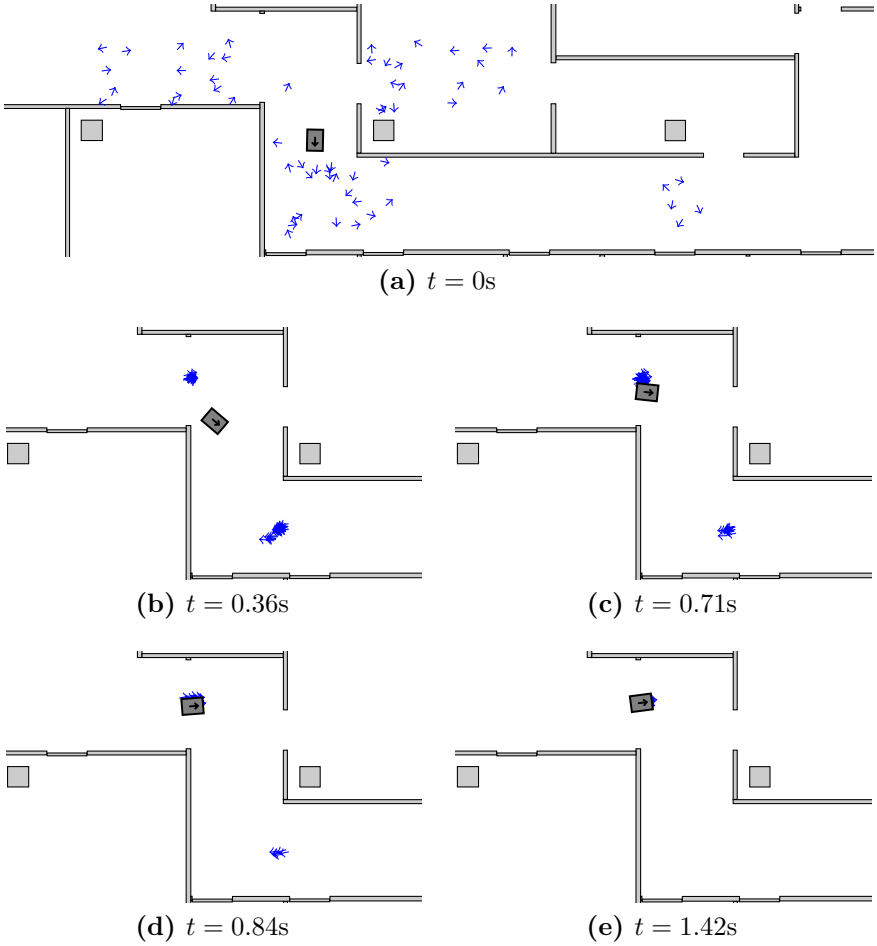
6.2.1. Global Localization using RSS Fingerprinting

Back in section 5.4, the localization of mobile robots in aware environments using the technique of received signal strength (RSS) fingerprinting has been discussed and first results have been presented. Also, the idea of using the rather coarse localization results of this method to assist a Monte Carlo localization algorithm has been outlined. In the scope of this application study, details and results of this combination are presented.

When the robot is activated, it is manually[1] instructed to read the previously created radio map from a file[2] and to start the localization process. During that process, the algorithm collects a number of frames (20 of them, like in the earlier experiment). As soon as the algorithm has finished, it returns a sorted list of candidate positions. The ten best candidates are fed to the Monte Carlo localization module. Assuming the localization uses a total of N particles to represent its belief, then for each candidate position, $N/10$ particles are uniformly initialized in a square (of $2 \times 2\text{m}^2$ area) around it. The robot's yaw angle is always uniformly initialized from all possible angles. Only by matching the local sensor information to the known map, the Monte Carlo localization can then quickly converge from many candidate positions to one determined position. For one example localization (out of many which have been tested during the application study), a sequence of the convergence process is shown in figure 6.1.

[1] There are no reasons to not automate this process in the future.

[2] In this experiment, the radio map created during the experiments from section 5.4 has been used. This shows that the localization approach is somehow robust against small disturbances, as the sensor nodes have been removed and re-installed in between those experiments.



(a) $t = 0$s

(b) $t = 0.36$s

(c) $t = 0.71$s

(d) $t = 0.84$s

(e) $t = 1.42$s

Figure 6.1.: Convergence of the Monte Carlo localization after being initialized with particles at candidate positions, as determined by the localization based on received signal strengths. The indicated robot position does not show the ground truth, but the mean position of all candidates.

It can be seen well, that at the beginning, the particles are broadly distributed among the candidate positions (although in the visualization, not all particles are drawn). As soon as new sensor measurements are integrated, the belief quickly converges to two possible locations and then finally to the correct robot pose.

Although it did not happen very often during the application study, it is of course possible that the localization using RSS fingerprinting does *not* provide the true position of the robot within the first ten candidate positions. In such a case, there may be no particles near the true position and thus the Monte Carlo localization fails to localize correctly. Usually, as soon as the robot moves and the map matching fails too badly, a re-localization should be striven for. This is currently not implemented, but would surely be a sensible improvement in the future.

6.2.2. Enhancement of the State Estimation by the Second Robot

In section 5.5, only a small experiment, with focus on the robustness in case of network failures, regarding the second robot has been presented. In the scope of the application study, the influence of the state estimation by the second robot has been examined a little further.

Exemplarily, to show the effect of the second robot to the state estimation, an occupancy graph mapping example has been chosen. During the experiment, the second robot (in reality) was placed at the south western corner of the hallway so that it was able to observe the two edges of the graph in that corner. See figure 6.2 for a photograph of the situation.

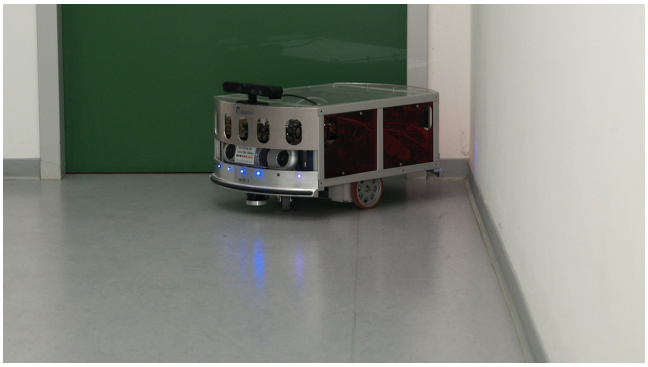
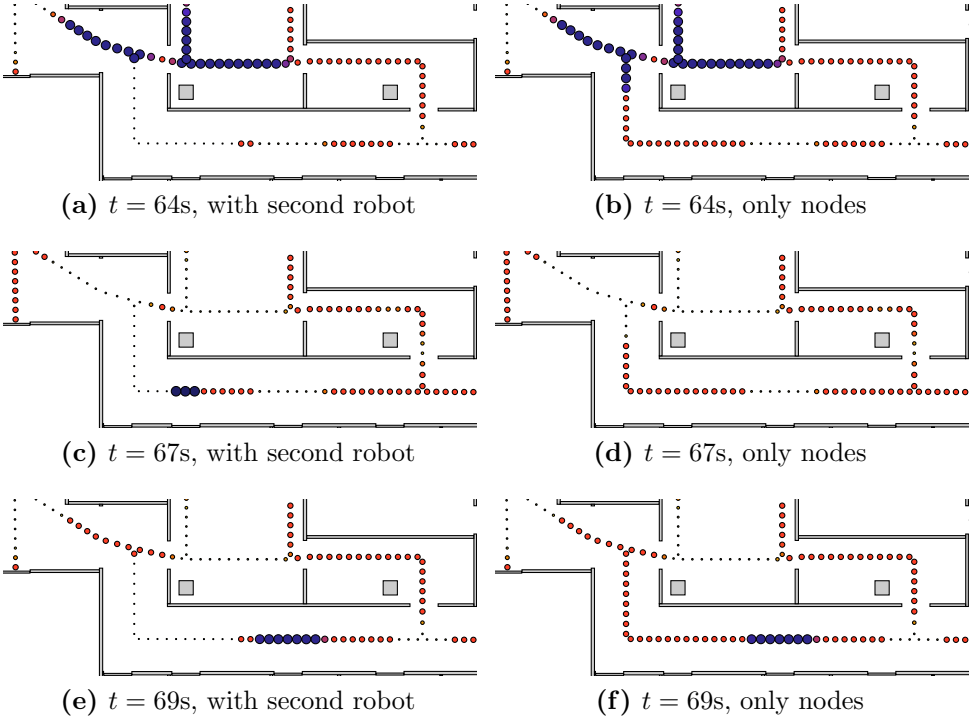


Figure 6.2.: The second ARTOS robot in the corner of the hallway (as photographed from a position near to the point A.

It should be noted that in this corner, the sensor coverage is rather poor (see figure A.10). Thus it was expected, that the sensor information from the second robot would improve the quality of the state estimation. During the experiment, a person (the author) was walking through the hallway. The sensor events (of the sensor nodes as well as the second robot) have been recorded for later analysis.

During the analysis, the occupancy graph mapping has once been performed *with* the external sensor data from the second robot and once *without* this data (i.e. only the information from the AmICA nodes was used in that case). For three points in time, the snapshots (with a direct visualization of the particles) is given in figure 6.3. For an easy comparison, the left side of this figure shows the situations with information from the external robot and the right side without.

At the first snapshot, at $t = 64$s, the person is sensed by the sensor with ID 255, this can be deduced from the highly weighted particles in that area. In case the data from the second robot is used, it can also be observed that a large amount of area that is uncovered by sensor nodes is correctly classified as "free". Interestingly, the data from the second robot even classifies some area that is also covered by the sensor node as "free", which allows a more exact estimation of the real position of the person. The



(a) $t = 64$s, with second robot **(b)** $t = 64$s, only nodes

(c) $t = 67$s, with second robot **(d)** $t = 67$s, only nodes

(e) $t = 69$s, with second robot **(f)** $t = 69$s, only nodes

Figure 6.3.: Comparison of a occupancy graph mapping sequence with and without incorporating the information from a second robot.

second snapshot shows a clear estimate of the position of the human in case the data from the second robot is used. In case it is not, the particles have their default weight, corresponding to "unknown", which represents the estimated situation well. Finally, in the third shown situation, the person triggers the sensor with ID 8, resulting in a similar estimation by both procedures. For a more detailed visualization of the sequence, please refer to section C.4.

6.2.3. The Influence of Prediction

Although the results from section 5.3 show that, given a large number of experiment runs, the effect of the prediction on the safety *can* be seen, the application study has shown that in practice, the effect may be less pronounced. In many typical situations, the results *with* and *without* prediction do not differ that much (which is a pity, because the prediction is a very costly process in terms of processing resources). There are however situations, in which there is a difference and that difference is also clearly visible when considering the safety of the path.

If there was activity west of the place A and the robot planned a path from B to A, then both methods of risk-assessing planning reproducibly gave distinct results.

If case the prediction is turned off, a path through the hallway was planned, see figure 6.4. This path is not particularly safe, because the motion model (or more clearly: the transition probabilities of the graph) state, that a person will likely *stay* in the hallway and *not* enter the meeting room (see figure A.9, which clearly shows this transition probability).

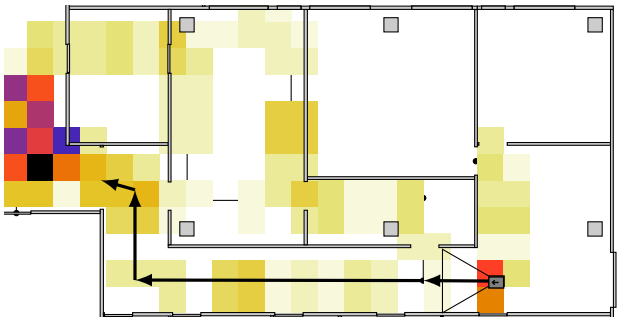


Figure 6.4.: Activity in the western part of the environment. The robot plans a path from B to A with risk assessment, but *without* prediction. The resulting path leads through the hallway (as indicated by the arrow).

If however, the prediction is turned on, the situation changes. Although the initial situation is very similar (as can be verified by comparing the beliefs at the time of planning), the prediction correctly takes care of the highly non-uniform transition probabilities at the entry to the meeting-room and outputs a path that guides the robot through the meeting room instead of through the hallway. This situation is depicted in figure 6.5.

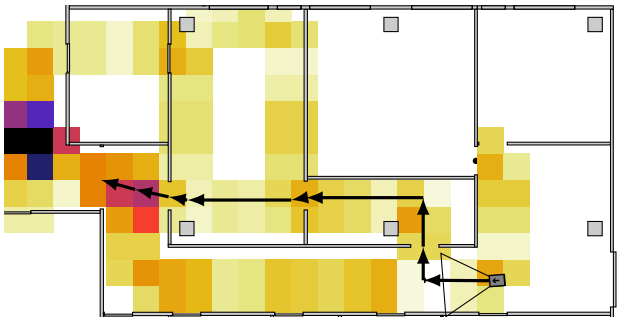


Figure 6.5.: Activity in the western part of the environment. The robot plans a path from B to A *with* prediction. The resulting path leads through the meeting room (as indicated by the arrow).

6.3. Discussion

The application study experiment has once more (after the single experiments, presented in the earlier chapters) shown the applicability of using sensor data from aware environments to improve mobile robot navigation. During the application study, all the methods that have previously only been analyzed separately, have been regarded in one large experiment. Received signal strength measurements from transmitters in the aware environment have been used to successfully assist the global localization process of a mobile robot through RSS fingerprinting methods.

The improvements to performance and safety due to usage of external sensor data have been thoroughly evaluated in the previous chapter, so no quantitative evaluations have been done during the application study experiment. However, the influence of a second robot as an additional sensor input to the state estimation process has been analyzed and it has been shown that it can be used to improve the estimation results, especially in areas that have poor coverage of ambient sensors. Additionally, the application study experiment has further researched the influence of the prediction during path planning, based on the estimated state.

7. Summary and Conclusion

7.1. Summary

This thesis motivates the use of sensor data, gathered from aware or smart environments to *simultaneously* optimize the three (usually opposing) design goals safety, low-cost and performance in the field of mobile robotics. Based on findings from the literature, several side theses which accompany this central thesis have been defined in section 3.3.

These theses state that the abovementioned goal of optimizing the three parameters should be reached by making use of *existing* infrastructure (i.e. technology that is already available in an aware environment), that the sensor instrumentation may be *sparse* and that the sensors itself should not invade the privacy of the inhabitants. Also, the people should not be required to actively participate for the system to work (i.e. they should not need to actively carry devices or behave differently than they normally would). Additionally, the system should be *fault-tolerant*, as aware technology is often unreliable.

In chapter 4, methods for estimating the state of the environment using *probabilistic* methods have been motivated, applied and developed. The use of probabilistic methods allows to cope with uncertainties and errors that are associated with the sensors and also allow to model the behavior of the people in the environment.

In chapters 5 and 6, several applications in the domain of mobile robotics have been developed, applied and analyzed that make use of data from an aware environment to optimize performance and safety of mobile robots. The applications include a virtual obstacle avoidance sensor that allows mobile robots to perceive the state of the environment beyond the ranges of their local sensor systems and an improved path planning algorithm that incorporates external information.

The former has shown to be able to increase the performance of a mobile robot by dynamically adapting its maximum permissible velocity depending on the estimated state of the environment. The path planning allows the robot to plan safe and efficient paths from start to goal positions. It has been evaluated in a version that aims to predict the evolution of the state during the path traversal and in one that only assesses the state at the time of planning. Depending on the scenario, both methods have shown to be able to improve safety according to a metric based on the distance between the robot and people in the environment. Further smaller experiments have evaluated an application that assists global robot localization using radio signals from

wireless sensor nodes in the environment and also the extensibility of the proposed system to further types of sensors.

7.2. Conclusion

Not only the application study experiment in the previous chapter, but also the smaller experiments conducted in the earlier chapters of this thesis, have thoroughly evaluated the methodologies proposed in this thesis. In this section, the concept for safety, cost-efficiency and performance, as proposed in form of the central thesis and the side thesis in section 3.3 is revisited with regard to the developed methods as well as the experimental evaluations of these methods.

> By consequently making use of *external* sensor data, it is possible to increase *cost-efficiency*, *safety* and *performance* of mobile robots.

From the beginning, external sensor data in form of sensor values from AmICA wireless sensor network nodes has been used to estimate the positions of people in the environment (see section 4.6, section 4.7 and section 4.8). Later, that external sensor data has been extended by additional information from other robots in the environment (see section 5.5). This data has been used to mainly increase safety as well as performance of a mobile robot during the path planning (see section 5.2 and section 5.3) and the path traversal (see section 5.1) phases of its tasks. Additionally, the radio signals from the AmICA nodes have been used to aid the robot in the process of global localization, i.e. to establish an initial localization estimation when powered up (see section 6.2.1).

Overall, these findings *can* be used to make future mobile robots cheaper by relying on more simple sensor systems. This has however not been shown in this thesis, as the effects presumably become only visible in a large scale adoption of such systems, which is obviously out of scope of this thesis.

> The *external* sensor data does not necessarily require the installation of *new* sensors in the environment. Instead, it shall be made use of *existing* infrastructure.

In the scope of this thesis, the AmICA wireless sensor network has been used to create aware out of traditional environments. This is of course at first contrary to the above claim, which states that *existing* infrastructure should be used. However, the important sensor of the AmICA nodes (that has been used for state estimation) is the passive infrared sensor. This sensor is very common in building automation, e.g. to activate interior or exterior lighting. Even in the main experimental environment (the office environment of the Robotics Research Lab), such sensors are already installed. The idea of using these existing sensors as input to the state estimation system is thus neither odd nor impossible. It is rather another step of integrating these already-existing sources into the overall system, which has to cope with questions of interoperability and standards. These aspects are not part of this thesis.

> The sensor instrumentation of the environment may be *sparse*.

With a maximum of only ten AmICA nodes (plus possibly a second robot as input to the state estimation), distributed through the whole environment (cf. figure A.10), the author claims to have created an example with sparse sensor instrumentation. The second meaning of "sparse" has also been adhered to in most of the experiments. Except for the experiment with the second robot as sensor input, the sensor information itself has been sparse (the PIR sensors only register "movement" or "no movement").

> The system shall be *fault-tolerant*, up to the *complete* failure of all external sensors.

Interestingly, certain sensors *did* fail in an unplanned manner during experiments. For some of these situations, it has *not* been noted by the author until analyzing the data off-line, as it did only minorly affect the performance of the proposed system. Additionally, for some experimental runs, nodes have been deliberately disabled, simulating sensor failures. In these examples, it has been analyzed how the performance of the system is affected (e.g. in section 5.1). Even during total failure of all external sensors, the system (the remaining part, i.e. the mobile robot) did not stop working, but continued with reduced performance. In case of an interrupted communication with the second robot, it has also been shown (in section 5.5), that the performance of the system degrades gracefully.

> The people in the environment shall not be required to actively participate for the system to work.

During the experiments, the people did not have to behave differently than they normally would have[1]. Also, they did not need to wear any badges or tags to be detected by the state estimators, as the passive infrared sensors detect people by their body heat. The principle of detection by the other robot as sensor is based on either laser rangefinders or an RGB-D camera, which also did not need any kind of participation by the people to detect them.

> The system shall not invade the privacy of inhabitants directly or make use of any sensors that would violate the privacy of inhabitants.

As stated earlier, mainly passive infrared sensors have been used as inputs to the state estimation system. Those sensors are able to detect moving people, but cannot identify them. Nor do they invade the privacy of the people in a way as e.g. cameras would. Still, it is possible to argue that the data from the passive infrared sensors can be used to infer information about e.g. the habits of people in the environment. For the second robot as sensor, the sensor information is much richer and thus much more privacy-invading, at least if the RGB-D camera is used. This example is however only one example to show the extensibility of the approach. There is no need to employ such rich sensors for the system to work as intended. At the beginning of this

[1] They, of course, had to adhere to the plans of the experiments, but the instructions did no tell the people to walk at unusual speeds or to intentionally trigger sensors.

thesis, in section 2.2.2, the importance of security in aware environments has been outlined. Unfortunately, the AmICA wireless sensor network platform does not offer any security at the time of writing. It can thus neither guarantee *confidentiality* nor *integrity* of the transmitted data. This data is very primitive, but if it is intercepted by an adversary, it can violate the privacy of the inhabitants of the aware environment nevertheless. If an adversary is able to transmit deliberately wrong information into the sensor network, this may of course lead to unsafe situations, especially in the case of virtual obstacle avoidance (as described in section 5.1).

To come to an overall conclusion: The goals defined at the beginning of this thesis have indeed been reached. Methods to reach these goals have been developed, analyzed and validated in experiments. The claimed increased cost-efficiency has been regarded in theory, as a practical evaluation would require significant further effort.

7.3. Future Work

There are still plenty of open ideas to improve the methods shown by this thesis. First of all, the author acknowledges the simplicity of the state-space as well as the motion model used in the experiments. To recap – the state space only consists of the current position of a human on the graph. It does not include any other information, like e.g. the person's speed or direction of travel. The "direction of travel" could even be extended further to e.g. the intent of the person.

Such extensions of the state space could certainly have the potential to improve the tracking accuracy and thus the overall performance of the system. However, it should also be noted that any additional dimension that is added to the state space requires the number of particles in the particle filter to be increased, which in turn also increases the computational complexity of the state estimation process. Additionally, new motion models for such extended state spaces would of course need information to act using this information.

This data, e.g. the association between motion patterns and intentions of the people, could be acquired using machine learning techniques. It is a debatable point whether all this could be done with the rather simple sensor inputs from AmICA sensor nodes. Then again, more complex sensor data might conflict with one of the central theses of this work, the use of sparse, anonymous, privacy-aware sensor data.

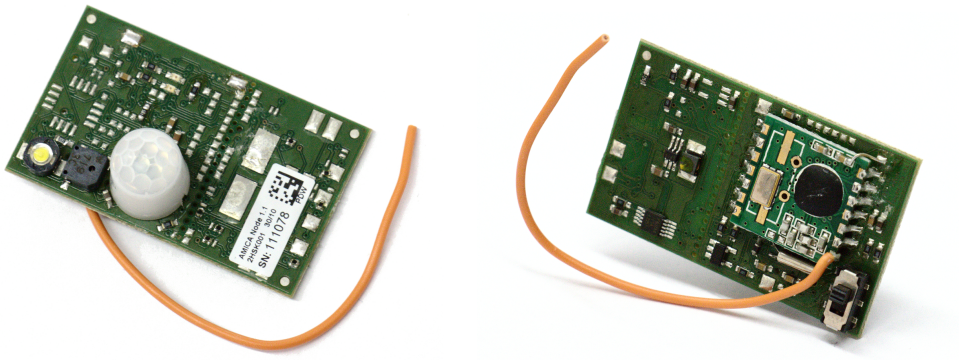
While the increase of cost-efficiency is theoretically possible in a large-scale view, it has already been noted that this increase has not been shown in this thesis. Mainly because of the requirement of having a large-scale robot deployment in an environment that is already aware to a certain degree, it has not been possible to practically validate this claim. Only the future itself (assuming further prevalence of aware and "smart" technology and the application of the synergy effect of robots in aware environments) can further explore and quantify the real cost savings.

A. Description of Experimental Platforms and Components

A.1. The AmICA Wireless Sensor Network Platform

A.1.1. Overview

The wireless sensors used throughout this work are nodes of the AmICA wireless sensor network platform (see figure A.1). They have been designed at the Microelectronic Systems Design Research Group, University of Kaiserslautern[1].



(a) Front view, showing LED, speaker and passive infrared sensor.

(b) Back view, showing the radio module.

Figure A.1.: Photographs of an AmICA node.

One of the design goals of the platform has been flexibility [Wille 10, 1.2]. It offers a variety of sensors and actuators that can be mounted on each node. It has been shown to be applicable (beyond this work) in Ambient Assisted Living scenarios [Wille 12] and for data acquisition in sports experiments [Wille 10, 4].

Each node uses an Atmel ATmega324P microcontroller running at 8MHz which allows low-power operation. Communication is realized using a Hope RF RFM12B radio module. The sensors and actuators that can be mounted directly on the printed circuit board are given in table A.1 and table A.2, respectively. Additionally to these

[1] `http://ems.eit.uni-kl.de/`

sensors, each node is also capable of measuring the value of its received signal strength indicator (RSSI) if it receives frames from others.

Sensor	Possible applications
Reed switch	State detection of doors or windows in conjunction with a magnet.
3D acceleration sensor	Detection of node movement and posture.
Passive infrared (PIR) sensor	Detection of moving warm objects like humans or animals.
Temperature sensor	Measure temperature between −55°C and 125°C.
Light dependent resistor	Measure brightness.

Table A.1.: Available sensors for the AmICA platform.

Actuator	Possible applications
Small speaker	Status signaling, general purpose sounds.
High-power LED	Status signaling, warning or orientation light.

Table A.2.: Available actuators for the AmICA platform.

A.1.2. Signal Processing

The nodes of the AmICA platform, as used in this thesis, do some rudimentary signal processing of the raw sensor data on board. This has the purpose of filtering and reducing the amount of data to be transmitted, but it also defines when data should be sent out over the network. For the processing of the passive infrared sensor data, this process is visualized in figure A.2. The raw signal is first low-pass filtered, as the built-in PIR sensors tend to generate many (binary) changes if an event is occurring (i.e. if someone is moving in front of the sensor). This filtered signal is then sent through an edge detector, which detects transitions between "movement" and "no movement" or vice versa. Either when this detector detects a change or when an internal timer event occurs, a "value change" event is created which triggers a transmission of the current value over the network. The period of this timer is set to 15s, so that, even if no event occurs, each node sends a value at least four times per minute.

The exact format of the frames that are sent by the nodes is of no importance for this thesis. It is only worth noting that each frame contains the unique identifier of the sending node, so that it can e.g. be used in the sensor model for state estimation or generally to identify the sender (e.g. for localization methods using the received signal strength indicators).

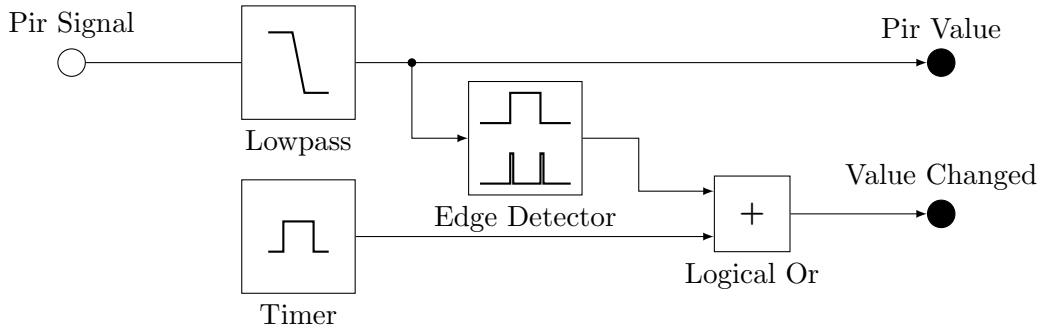


Figure A.2.: Schema of the local signal processing of the passive infrared sensor's values on each AmICA node. Originally created in a similar version for [Arndt 11a].

A.1.3. Security

Unfortunately, the AmICA nodes (at the time of writing) do not employ any cryptographic measures to ensure either confidentiality or integrity. In short: security measures are not present. The reader may argue that a very simple solution to this problem might be to just have a single shared key that can be used to encrypt all the messages using a state of the art encryption algorithm, e.g. AES.

While this might improve the integrity of the network dramatically, and mitigate the number of possible attacks, there are many more sophisticated attacks that cannot be avoided using only ordinary encryption. A major threat to encryption are always so called side channel attacks. Side channel attacks have e.g. been successfully used against encryption in voice over IP (VoIP) communication [Wright 08]. A similar attack of a (hypothetical) encryption in the application of AmICA node in this thesis could look as follows: Nodes periodically transmit the state of their sensors, but also (out of schedule) if there are changes to the sensor states. An attacker could easily detect these transmissions and infer "activity" at these moments of time. This simple thought-experiment could easily break the allegedly present confidentiality in case of a simple encryption on the network layer.

A.2. The Mobile Robot Artos

A.2.1. Overview

As a demonstrator and exemplary robot to demonstrate the feasibility of the proposed methodologies, the mobile robot Artos is used (see figure A.3). This robot has initially been designed for Assisted Living scenarios [Armbrust 07, Koch 08, Berns 10], but as a multi-purpose mobile indoor robot, it is of course not limited to such applications.

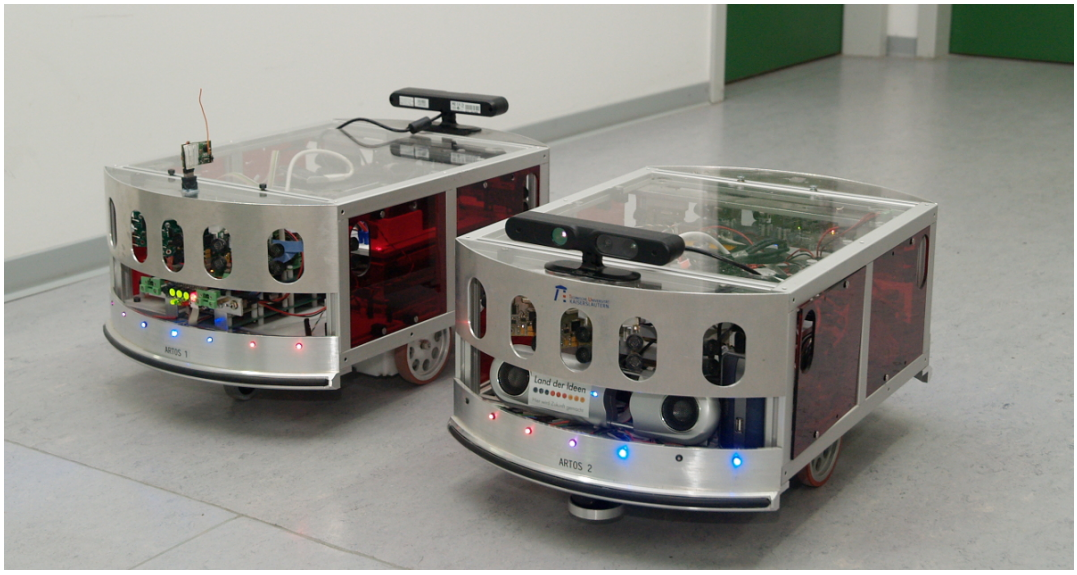


Figure A.3.: The two mobile robots of the type ARTOS that have been used throughout this thesis. At the rear of the left robot, the receiver for the AmICA wireless sensor network platform can be seen.

A.2.2. Mechanics

The mobile robot ARTOS has a length of approximately 50cm while being approximately 30cm wide. With lead batteries mounted, it weighs about 20kg. However, these have recently been replaced by batteries based on lithium-ion polymer technology, reducing the weight of the robot significantly to about 13kg (including the battery). The main chassis is constructed using aluminum profiles with custom-made parts for the front and back of the robot. The drive system consists of two driven wheels with a differential drive kinematic and two passive wheels to support the chassis on the back and on the front.

A.2.3. Hardware

The hardware of the robot consist of several levels of components, see figure A.4. On the high level, there is a small computer with an x86-architecture mounted which is responsible for running an operating system and performing all the high-level processing. At the beginning of the work on this thesis, this was a Mini-ITX mainboard with 4 GiB of RAM, Intel Core Duo U2500 processor (clocked at 1.20GHz), IEEE 802.11abg WiFi card (Intel 2915ABG) and a 2.5 inch notebook hard disk drive. Later, this hardware has been replaced by a (at the time of writing this section) more up-to-date computer: an Intel D54250WYK NUC (Next Unit of Computation) with

a low-power Intel Core i5-4250U processor, 8 GiB of RAM, IEEE 802.11n/ac (Intel Dual Band Wireless-AC 7260) and an mSATA solid state disk.

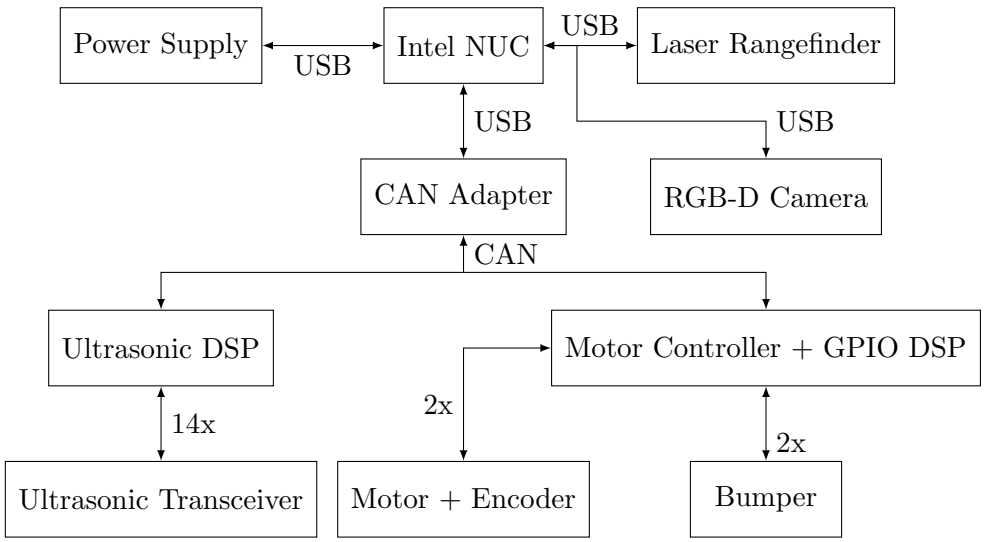


Figure A.4.: Overview of Artos' hardware.

Directly connected to this computer are some of the sensors, like a laser rangefinder and an RGB-D camera (the exact types are listed later in section A.2.4, when the sensor systems are discussed). Additionally, some peripheral devices such as a gamepad (for manual control), text displays and also the power supply of the robot are connected via USB. For brevity, not all of them are shown in the figure.

A USB to CAN (Controller Area Network) adapter establishes a connection between the high-level and the low-level electronic components. Those components mainly consist of two embedded Digital Signal Processors (DSPs), which accomplish several low-level tasks. One DSP handles the operation of a total of 14 ultrasonic sensors (see also section A.2.4 about the sensor systems of the robot) to be able to measure distances to obstacles around the robot. The other DSP's main task is the closed-loop control of the drive motors. It reads the wheel encoder values and controls the motors using power electronics. Additionally, some of its GPIO (General-Purpose Input/Output) pins are used to connect to the bumpers at front and rear of the robot that are triggered in case of collisions.

A.2.4. Sensor Systems

The robot Artos, as it is used during the experiments in this thesis has several sensors mounted. The types of the sensors are diverse with different advantages, disadvantages and sensing ranges, see the overview in figure A.5.

The total of 14 ultrasonic (seven at the rear and seven at the front of the robot) have already been mentioned. They are controlled using one of the DSPs and can

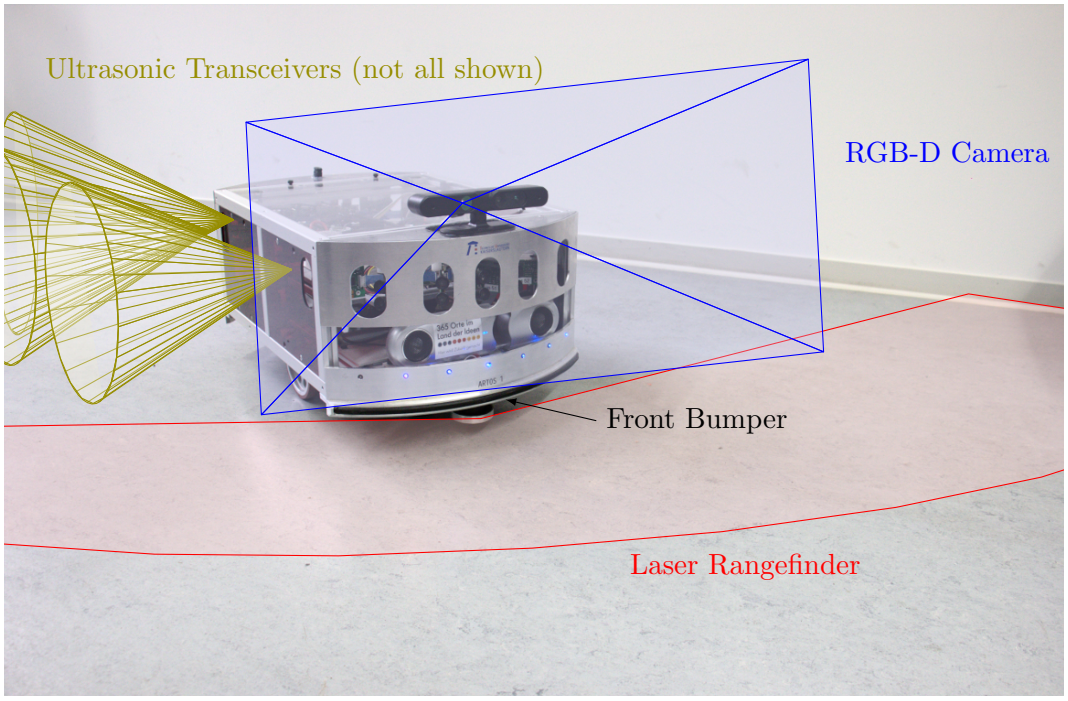


Figure A.5.: Overview of Artos' sensor systems. The indicated sensor ranges are *not* to scale.

be accessed through the CAN bus. Their range is rather short and due to the measurement principle, the sensor values are highly superimposed with noise and many of the measurements are erroneous. For this reason, their importance in this work is marginal. The rear and front bumpers can be seen as a "last resort" to stop the robot, if it has already collided with an object either in front or behind the robot. During normal operation, they should never be needed to stop the robot.

The laser rangefinder in the front is the model URG-04LX by Hokuyo. It has an opening angle of 240° and a range of up to 4m. It is a source of very precise distance measurements that are in this thesis mainly used for the Monte Carlo localization. The RGB-D camera mounted on top of the robot is an Asus Xtion Pro Live device. It employs a sensor by Primesense which is targeted at the consumer market, and thus is not very expensive. It can be seen as a low-cost sensor. If the sensor is used indoors (which is the case with Artos, which is a pure indoor robot), the range can easily surpass that of the laser rangefinder, but unfortunately the field of view of the camera is very small, compared to the laser rangefinder (58° horizontally, 45° vertically[2]). In earlier times, the robot Artos was also equipped with a radio-frequency

[2]According to the specification provided by the manufacturer.

identification (RFID) sensor which had been used for localization in earlier works of other authors. This sensor was *not* mounted at the robot during the experiments. Also, the environment was not equipped with any RFID tags for localization or other purposes. For the experiments in this thesis, the robot was equipped with a receiver for the AmICA wireless sensor network, to be able to receive the sensor data directly on the robot (see also figure A.3).

A.2.5. Control System

This subsection briefly describes the control system of the mobile robot Artos. During this thesis, the control software of the mobile robot Artos has been migrated from the old *Modular Controller Framework 2 (MCA2)* framework to the much more powerful Finroc framework. Both are frameworks that aim to support the developers and researchers of robotic systems with their work. This is done by providing library code and means for structuring and interconnecting large robot control system, consisting of many components.

The Finroc framework has been designed to promote software quality in robotics and ease the development of complex robotic systems [Reichardt 13b, Reichardt 13a]. Finroc offers methods to hierarchically structure robotic systems using components with clearly defined interfaces. Using plugins, it also allows to interconnect components using different protocols and technologies, e.g. using the network protocol TCP/IP. Thanks to a powerful component defined visualization, Finroc allows easy analysis and debugging of single components and whole robotic systems [Reichardt 14]. The framework has also been shown to work on very simple architectures, like a softcore implemented on an FPGA without operating system [Schuetz 14] and to be easily interoperable with other popular robotic frameworks, such as the *Robot Operating System (ROS)* [Arndt 13a].

The actual control system of the mobile robot Artos has been implemented using the Finroc framework. A schematic view of the control architecture is shown in figure A.6. Starting with the bottom of the illustration, it is possible to either interface the real hardware or the simulated robot within a simulation environment (see also below, section A.4 regarding the implementation of the simulation).

At the top of the control hierarchy, there are application-specific components that may be instantiated depending on the tasks for which the robot is employed. This could be the "transportation task" that is used for some of the experiments in the main chapters, some exploration behaviors that aim to explore the environment and e.g. create a map of it or something completely different (like an Assisted Living scenario).

Regarding the layer between the application-specific components and the abstraction of hardware or simulation, this is where the interesting and most important components of the robot reside. First of all, incoming sensor data is processed. Raw, noisy values are filtered (especially those of the unreliable ultrasonic sensors) and sensor

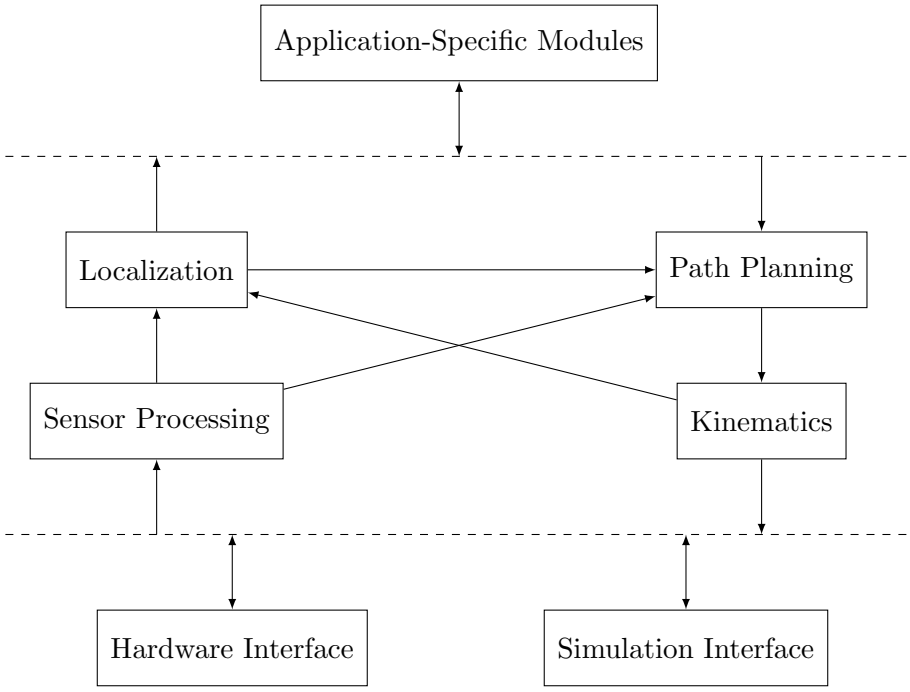


Figure A.6.: Overview of ARTOS' software control system.

information from several sources are brought into the same coordinate system (the robot coordinate system) and are fused together. This data is then used for localization as well as for (local) path planning.

The localization components consist of simple dead reckoning using wheel encoders and additionally, a more sophisticated particle filter localization approach (see section 2.1.2.3 about Monte Carlo localization) provided by the *Mobile Robot Programming Toolkit (MRPT)*.

The path planning components encapsulate planners on a global as well as on a local scale[3]. They comprise the algorithms that have been introduced in the main chapters (on a global planning scale) and a local navigator which is also provided by MRPT. In case of the virtual obstacle avoidance sensors, the output of the local planner is filtered by a module which limits the permissible velocity of the robot, when needed.

A.3. Offices of the Robotics Research Lab

This section documents the office environment of the Robotics Research Lab which has been used for all important experiments and evaluations.

[3] As defined in section 2.1.3

A.3.1. Floor Plan

Figure A.7 shows the floor plan of the relevant parts of the Robotics Research Lab. Also given in this figure are the locations of three important rooms/places that are referenced in the main chapters – the meeting room, the kitchen, the hallway as well as the places A and B between which the robot navigates. Figure A.8 contains photographs showing these relevant locations in reality.

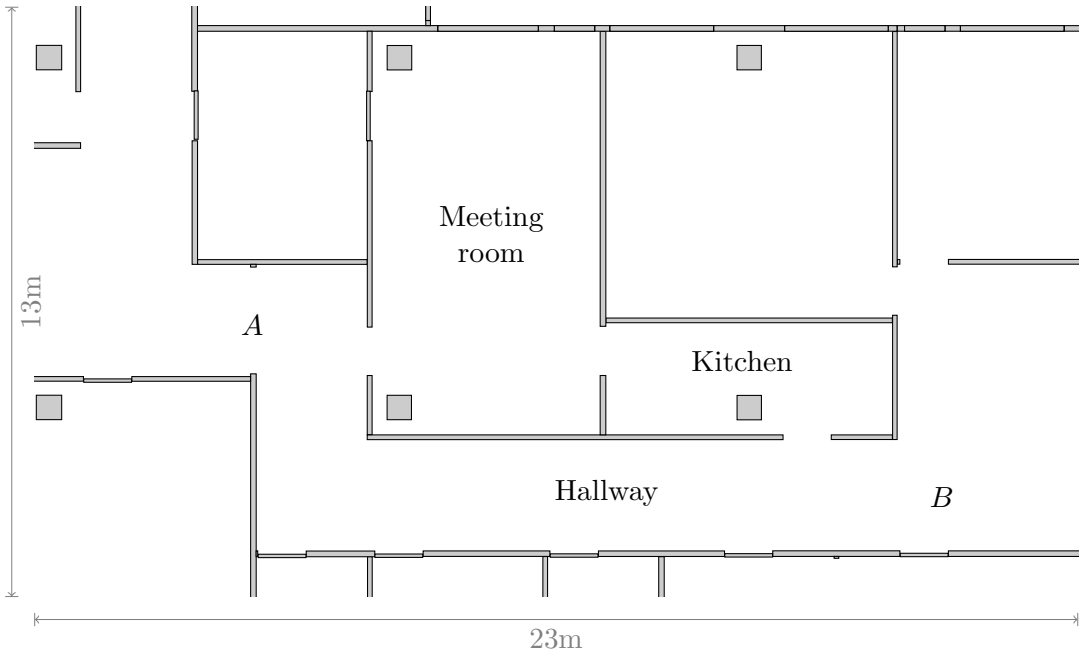


Figure A.7.: Floor plan of the relevant part of the premises of the Robotics Research Lab, University of Kaiserslautern.

A.3.2. Graph Layout

Figure A.9 shows the graph which is used for state estimation as well as for robot path planning. To visualize the Cartesian positions of the graph's vertices, it is drawn on top of the already known floor plan.

This figure also visualizes the non-uniform transition probabilities for traversing from one edge of the graph to another (as described in section 4.3). For vertices that do not have an arrow originating, the transition probabilities are uniformly distributed among all out-edges of that vertex. If e.g. a vertex has two out-edges[4] and the transition probabilities are distributed uniformly, then each edge has a probability of 0.5 of being chosen.

[4]Which is equivalent to two adjacent vertices, as the graph is undirected.

(a) Part of the hallway, showing point A at the center of the (opened) door.

(b) Part of the hallway, showing point B.

(c) The kitchen as seen from the door to the meeting room.

(d) The meeting room as seen from the north-western corner.

Figure A.8.: Photographs of the relevant locations of the environment marked in figure A.7.

In case a vertex has an arrow originating from it in the illustration, this means that the probability of choosing the out-edge along the arrow is given by the number written below. The "remaining" probability is then uniformly distributed to all other out-edges that have no transition probability explicitly set.

In the above example, the specified transition probabilities can be interpreted as follows: If a person is currently situated in the meeting room (cf. figure A.7), then there is a rather high probability that the person will either stay in the meeting room or leave it through the door to the kitchen. Similarly, the arrow between kitchen and meeting room states that if a person is present in the kitchen, it is quite unlikely that he or she enters the meeting room. It is much more likely that the person will leave the kitchen afterwards in direction of the hallway. The probabilities attached to vertices in the hallway aim to characterize the idea that the majority of the people stays in the hallway (and that they do seldom enter the meeting room), but also that they *might* enter the kitchen on their way.

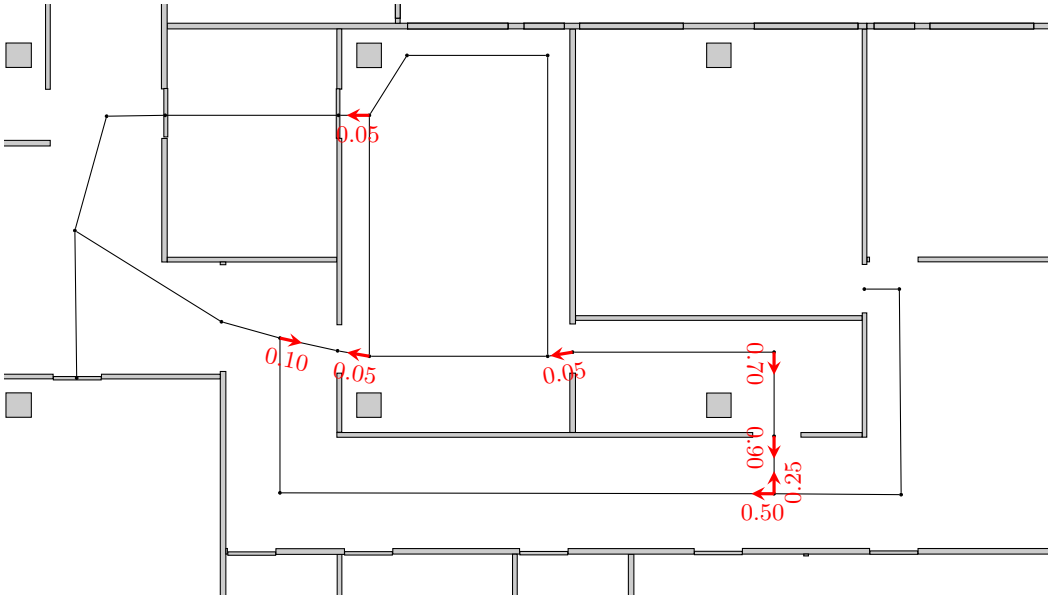


Figure A.9.: Graph used for state estimation and robot path planning in the experiments within the office environment of the Robotics Research Lab.

A.3.3. Sensor Model

A visualization of the sensor positions and the resulting sensor model in this environment is given in figure A.10, where the information has been overlayed on top of the floor plan and the graph. The identifiers of the sensor nodes have also been included, as some nodes are referenced in the main text.

This figure clearly shows that this kind of sensor model (as described in detail in section 4.5) only works with the graph structure, as it completely ignores the walls and other obstacles in the environment. According to the Cartesian visualization of the model, it can "see" through walls, which is obviously not true. As however only positions at the edges of the graph need to be correctly handled by the model, the reader may verify using the figure that in that case, the models are correct[5].

For the parameters of the single sensor nodes, that are *not* directly visible in the visualization[6], the following default values have been used: The mounting height is set to $h_m = 2\text{m}$, the sensing range to $r_s = 5\text{m}$. The vertical opening angle to $\alpha_v = 82°$, the horizontal opening angle to $\alpha_h = 100°$ and the default vertical mounting angle to $\beta = -60°$, i.e. facing slightly downwards to the ground. The probabilities for true positives and true negatives have both been set to $p_{tp} = p_{tn} = 0.9$. These are the

[5] For very dense graph layouts, there may be situations in which a sensor model incorrectly covers (for that sensor) invisible parts of edges. Such cases may require either re-arrangement of the nodes or an even more sophisticated sensor model that handles occlusions.

[6] The mounting position (x, y) and the horizontal mounting angle α are visible in the picture.

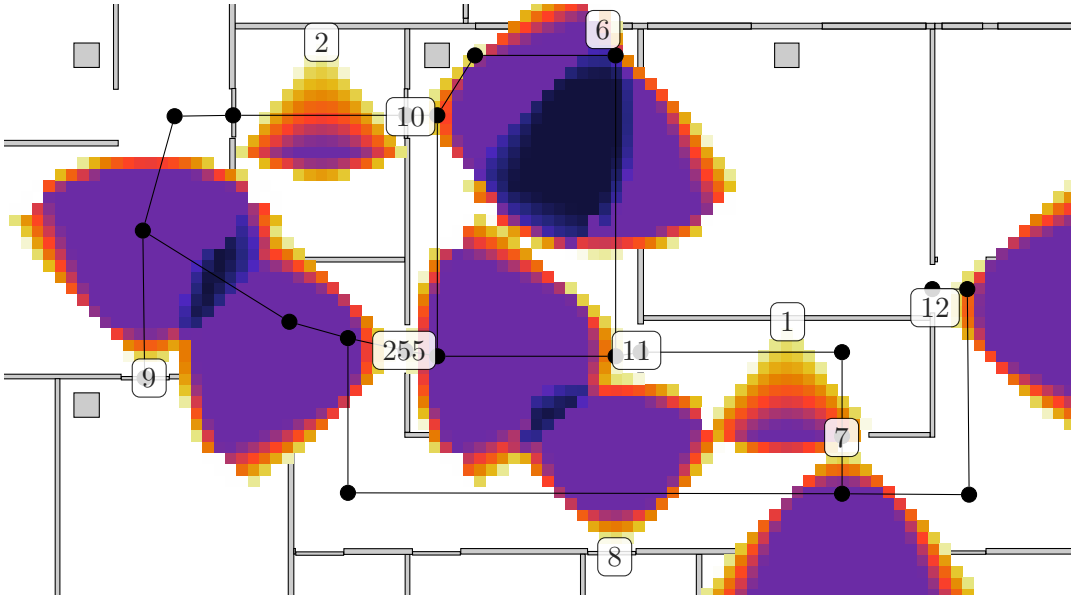


Figure A.10.: Sensor positions (with identifiers of the nodes) and visualization of marginalized inverse sensor model used in the experiments within the office environment of the Robotics Research Lab.

default values for a node mounted on a door frame, see figure A.11 for an example showing the mounting of the node with identifier 12.

For some sensor nodes, these default values had to be changed to other values, see table A.3. Unlike the other nodes, the nodes with identifiers 1 and 2 are not mounted on door frames, but on a whiteboard and a cabinet, respectively. These have a different mounting height and also a different vertical mounting angle (not tilted towards the ground). Additionally, to not match particles on wrong parts of the graph, due to walls in the environment, their sensing range has been artificially decreased to 3m. For the same reason, the range of the node with identifier 8 has also been slightly decreased.

Node ID	**Mounting height** h_m [m]	**Sensing range** r_s [m]	**Vertical mounting angle** β [°]
2	1.5	3	−90
1	1.5	3	−90
8	2	4	−60

Table A.3.: Non-default parameters of the single sensor nodes used in for the sensor model of the office environment of the Robotics Research Lab.

Figure A.11.: AmICA node mounted on a door frame.

A.3.4. Motion Model

As a motion model, the Brownian motion introduced in section 4.4 has been used. In the motion model, the speed of a person (in m/s) is uniformly drawn from the interval $[-1.5, 1.5]$. The sign of the value decides in which direction on the edge of the graph the person moves. This speed is then multiplied with the time since the last update of the state estimator to make the person move that distance.

A.4. The Simulation Environment

A.4.1. Overview

The mobile robot Artos as well as the AmICA nodes have not only been used in reality, but also within a simulation environment. There are several reasons, why this has been necessary and why not only the real robot has been used. First of all, experiments in robotics tend to be very time consuming, especially if they involve larger areas and the participation of human beings. The time to set up the experiment, to create controlled conditions does often constitute a large amount of the total time spent to conduct an experiment.

Another problem is the determination of grounds truths, whenever some estimated value is to be compared with the true value, e.g. in tracking experiments and evaluations. Manually recording the ground truth can also be a very laborious task, unless there are special systems used that can in some way capture the ground truth (e.g. motion capturing systems to record human or robot positions during experiments).

Altogether, this makes gathering large data sets through experiments in reality a tough problem.

For this reason, from the very beginning of this work, not only the real robot has been used, but also a detailed simulation environment has been set up and used. This simulation environment is based on the SimVis3D framework [Wettach 10], which has been used to simulate many aspects of robotics in the past.

A.4.2. The Environments

All environments in which simulation experiments have been conducted, have been modeled in the simulation environment. Originating from a (two-dimensional) plan of the environment, the automatic creation of three-dimensional models that can be used within SimVis3D is possible. The only environment which is relevant for this thesis is the office environment of the Robotics Research Lab. A rendering of the three-dimensional model of the environment in the SimVis3D simulation tool is shown in figure A.12.

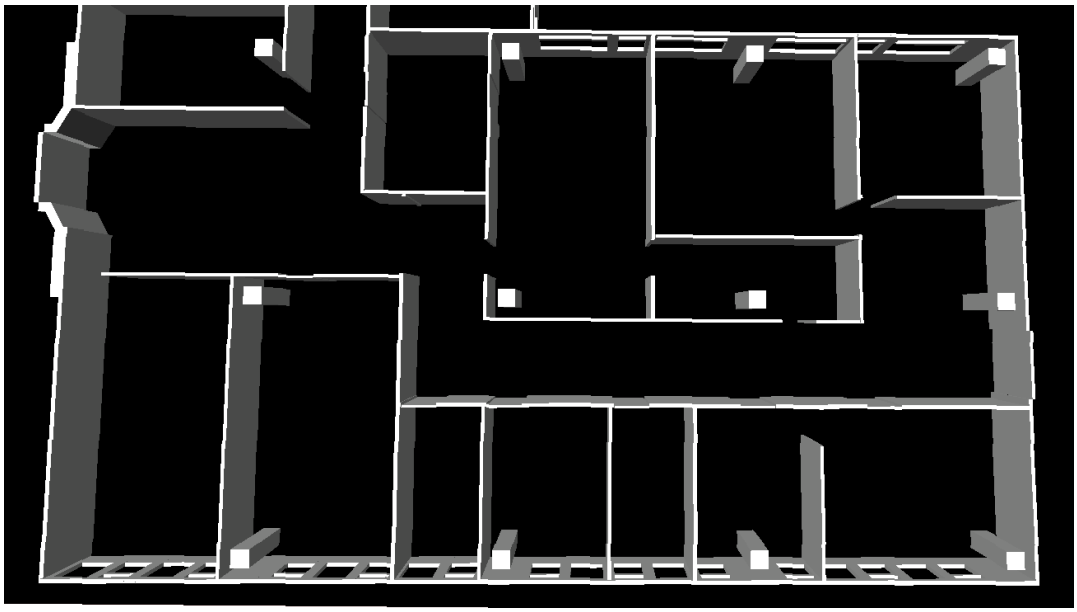


Figure A.12.: View of the simulated office environment of the Robotics Research Lab (without furniture).

A.4.3. The Robot

The mobile robot Artos (see section A.2) with all of its relevant sensor systems has also been modeled in the simulation environment. The robot's simulated differential

drive system uses the same parameters (distance between wheels, wheel positions etc.) as the real robot. The contact of the wheels with the ground is simulated using the physics engine Newton within the SimVis3D framework. Thanks to this physics simulation, the simulated wheels experience similar effects as the real ones, like e.g. slip in either the turning direction or sideways.

All the important sensors of the real robot are also available with the simulated robot. The wheel encoders are emulated by measuring the velocities of the wheels using the physics engine. The laser rangefinder, the depth image of the RGB-D camera, the ultrasonic sensors as well as the front and rear bumpers are all modeled using simulated distance sensors that measure distances between the sensor position and other objects in the 3D scene. The RGB image of the RGB-D camera is also available through offscreen rendering of the 3D scene with the viewpoint of the camera. A rendered image of the mobile robot Artos in the simulation environment is given in figure A.13.

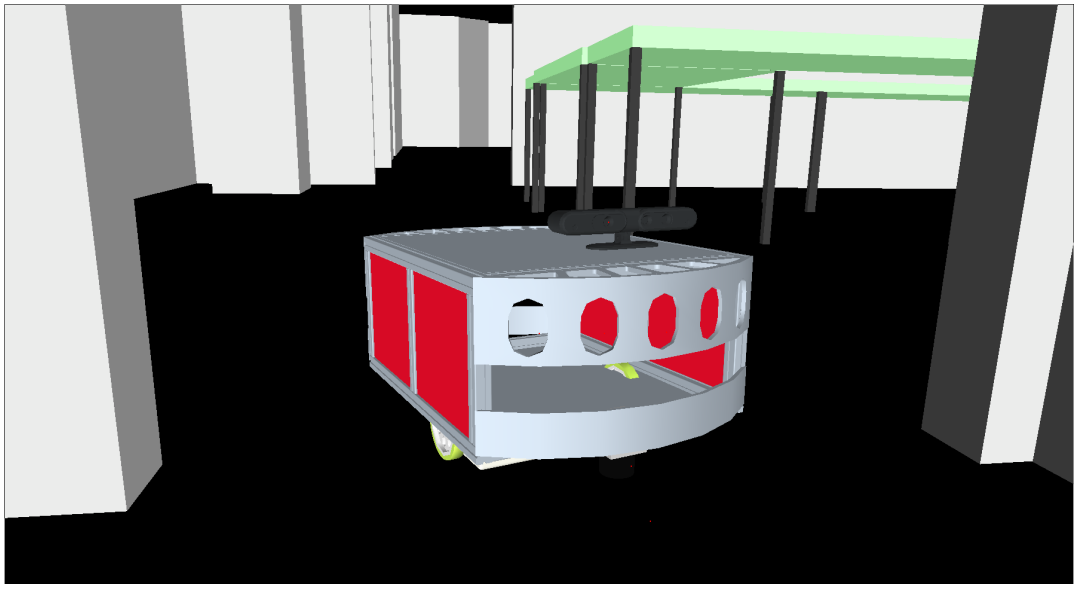


Figure A.13.: View of the simulated robot Artos showing part of the simulated meeting room.

A.4.4. The AmICA Nodes

Apart from the robot, also the wireless sensor nodes of the AmICA wireless sensor network (see section A.1) have been modeled in the simulation environment. The radio communication and also the most important sensor, the passive infrared sensor have been dealt with in the simulation. Most of the aspects that follow in the next sections have been developed in the scope of an internal report [Arndt 11b], and some parts have already been published in [Arndt 12b, 3.2].

A.4.4.1. Radio Communication

Radio communication is complex. If signals are transmitted at radio frequencies, they behave unintuitively and are subject to many different phenomenons that make predicting their exact behavior very hard. These have been discussed previously in section 5.4.2. The exact simulation of the propagation of radio waves is therefore often infeasible, but luckily also only required in rare special applications. Instead, appropriate models of radio communication are used that model only the aspects that are relevant in the application scenario. In the literature, there exist methods and tools to simulate radio communication at different levels of complexity. See e.g. [Korkalainen 09] for an overview and a comparison of different existing tools.

For the simulation of the AmICA wireless sensor nodes, only the concurrent medium access and the resulting destruction of frames when there are multiple transmitters at once, is simulated. When the transmissions of two transmitters (that are sufficiently close to each other) overlap, the radio frames may get corrupted. This is under the assumption that both transmit on the same frequency and that there are no *Code Division Multiplexing (CDM)* schemas in use that allow to recover the original data. Both of these assumptions are true for the AmICA wireless sensor nodes.

To detect such a collision in a simulated transmission medium, the time of the start of the transmission for each transmitter t_{tx_i} and their length (which is assumed to be a constant) t_{trans} must be known. Whether a collision occurs can then easily be checked by pairwise comparison of the timestamps t_{tx_i} and t_{tx_j} for the pair of transmitters (i, j). If the start of the later transmission ($t_{\mathrm{tx}_j} > t_{\mathrm{tx}_i}$) falls into the interval occupied by the earlier transmission, then a collision occurs. In case of $t_{\mathrm{tx}_j} < t_{\mathrm{tx}_i} + t_{\mathrm{trans}}$, a collision occurs and both frames are discarded by the simulation framework. This process is visualized by two examples in figure A.14.

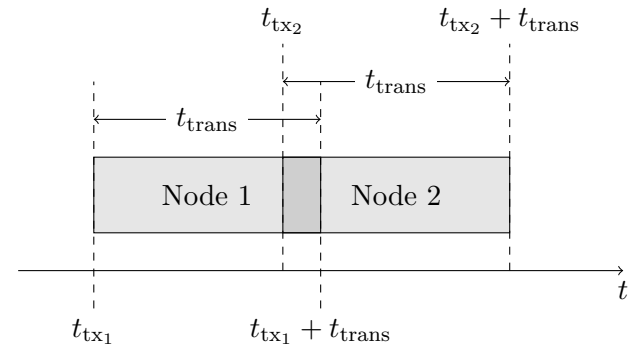
A.4.4.2. Passive Infrared Sensor

Accurate simulation of passive infrared sensors is very important for the work presented in this thesis, as it is the main sensor input for the tracking of people in the aware environment. Additionally, the physical property that underlies the measurement principle of the sensor (the propagation of thermal radiation) is not a property that is typically simulated in simulation frameworks or physics engines.

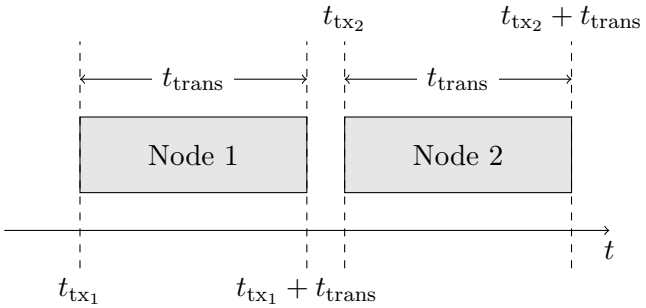
The measurement principle of PIR sensors is simple. A specially designed lens collects thermal radiation in the far infrared spectrum and focuses it on a pyroelectric sensor that is able to detect the radiation. If now a warm[7] object (such as a human or an animal) passes through the observed region, the sensor can detect that change and issue a “motion” event.

Regarding the simulation of such sensors, a problem arises. A very simple approach would be to use a low resolution offscreen renderer with a similar field of view as the

[7] Compared to the surroundings.



(a) Bad case: $t_{tx_2} < t_{tx_1} + t_{trans}$, a collision occurs on the medium



(b) Good case: $t_{tx_2} \not< t_{tx_1} + t_{trans}$, no collision occurs

Figure A.14.: Collision checking on the simulated shared medium. Figure originally created for an internal report [Arndt 11b] and first published similarly in [Arndt 12b].

PIR sensor's lens and then issue a "motion" event if and only if the frame changes significantly over time. However, this completely neglects the fact that only *warm* moving objects trigger the sensors. For this reason, the SimVis3D framework has been modified to attach temperature attributes to objects in the 3D scene.

Using these attributes, it is then possible to switch the whole scene to a "false color" thermal image representation and render it using a standard offscreen renderer. After all the thermal images have been rendered, the scene is switched back to the original colors. This whole process works reasonably well, because the far infrared radiation behaves similar as visible light (this principle is also used by thermal cameras that create false-color thermal images). An example of a scene rendered in visible as well as in thermal representation is given in figure A.15, see also [Arndt 13b, 4.3] for additional examples.

After the thermal image of a simulated PIR sensor has been acquired, it must be further processed to detect the motion events. As described earlier, the original PIR sensors react on *changes* of temperature, so this has to be emulated in the simulation environment. The data processing with the thermal images works in

(a) Normal colors **(b)** Thermal image

Figure A.15.: A 3D model of a woman within SimVis3D, rendered in visible light texture as well as in thermal image texture. The original (true color) model is licensed under the terms of the CC-BY-SA by Jeffrey Mills, ADL Co-Laboratory. The derived *thermal* texture is thus also licensed under the same terms.

three major steps. First, the images (pixels representing temperature values) are differentiated with respect to time ($\mathrm{d}T/\mathrm{d}t$), afterwards a temperature threshold is applied that sets temperature-differences that are too small to zero, others to one. In the implementation, this step is realized by calculating the difference of two consecutive image frames and then thresholding the image. The second step integrates the result with respect to space ($\int \mathrm{d}s$), to measure the overall change between the images. This is implemented by averaging all the pixels of the image. The last step is another application of a threshold to the scalar value of the previous step. Only if the value is large enough (i.e. iff *enough* pixels have a sufficient temperature change), a "motion" signal is asserted. This process is schematically represented in figure A.16.

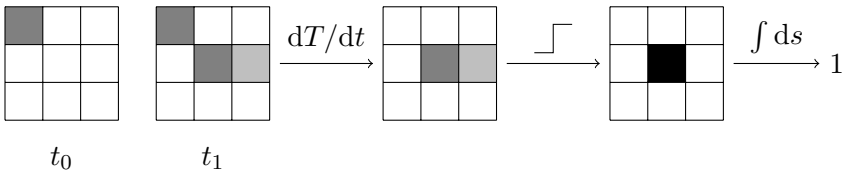


Figure A.16.: Data processing during simulation of PIR sensors. A similar version has been originally published in [Arndt 13b].

B. Derivation of Formulas and Proofs

B.1. Nyquist-Shannon Theorem applied to Mapping from Graph to Grid

To be able to correctly map the graph onto a grid, the relation between the width of the edges and the step size of the grid needs to fulfill some requirements. Interestingly, these requirements can be obtained by looking at the field of sampling theory, more specifically by using the Nyquist-Shannon sampling theorem. The Nyquist-Shannon theorem (which can be found in many textbooks about digital signal processing, e.g. [Williston 09, 1.2.2]) states that a signal of a maximum frequency f needs to be sampled with a *sampling frequency* f_s so that $f_s > 2f$ holds. If that inequality is violated, aliasing effects might be the results, which can render the sampled signal unusable. In many contexts (including this), it makes more sense to speak about the minimum period T of the signal and the *sampling period* T_s instead of frequencies, which are just the reciprocals of the corresponding frequencies, so:

$$f_s > 2f \tag{B.1}$$

$$\frac{1}{T_s} > 2\frac{1}{T} \tag{B.2}$$

$$T > 2T_s \tag{B.3}$$

Now it needs to be clarified how the problem of mapping a belief in form of particles on a graph to an occupancy grid is related to sampling. The sampling works by looking at particles (which are extended from points to line segments that represent their width on their edges) and then determining one or more cells of the grid to which these geometrically extended particles map to. So what is actually happening is that the cells of the grid are sampled by the particles. This might be unintuitive at first, but that is what is actually happening. For this reason, the periods of the theorem do correspond to the sizes in the problem in the following way: The sampling period corresponds to the width of the edges and the signal period corresponds to *twice* the step width of the grid. This gives the first result:

$$2 \cdot \text{step width of the grid} > 2 \cdot \text{width of the edges} \tag{B.4}$$

$$\text{step width of the grid} > \text{width of the edges} \tag{B.5}$$

One may now ask the question why the signal period corresponds to *twice* the step width of the grid. The answer is best answered using the small sketch in figure B.1. If a sine wave is used to represent the signal, the first half of the period maps to the first grid cell and the other half maps to the second grid cell.

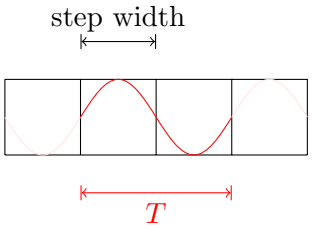


Figure B.1.: Illustration of the signal period which is equal to twice the step width of the occupancy grid map.

As already stated in the main text, it is possible to unambiguously sample two grid cells (which may be identical) at the beginning and at the end of the line segment representing a given particle, if the above condition is fulfilled. This represents the case $n = 1$ (where n is the number of sampling points at each side of the edge). If the value of n is increased, there will be not only two, but $2n$ samples in total for a single particle. This increases the sampling frequency to nf_s and will thus decrease the sampling period to T_s/n. When this new sampling period is inserted, the equation can be rewritten as:

$$2 \cdot \text{step width of the grid} > \frac{2}{n} \cdot \text{width of the edges} \tag{B.6}$$

$$\text{step width of the grid} > \frac{1}{n} \cdot \text{width of the edges} \tag{B.7}$$

If (B.7) is solved for n, the above inequality is obtained:

$$\text{step width of the grid} > \frac{1}{n} \cdot \text{width of the edges} \tag{B.8}$$

$$n \cdot \text{step width of the grid} > \text{width of the edges} \tag{B.9}$$

$$n > \frac{\text{width of the edges}}{\text{step width of the grid}} \tag{B.10}$$

□

B.2. Braking Distance and Safe Velocity

The derivations of the braking distance and the safe velocities are based on the following two well-known formulas from mechanics that define the relations between acceleration a, velocity v, displacement x and time t:

$$v = \int a \, \mathrm{d}t \tag{B.11}$$

$$x = \int v \, \mathrm{d}t \tag{B.12}$$

In case of constant acceleration (which is assumed for the brake deceleration), (B.11) can be written as (B.13).

$$v = a \cdot t \tag{B.13}$$

Inserting (B.13) into (B.12) and integrating yields a function to compute the displacement x in terms of t in (B.15):

$$x = \int a \cdot t \, \mathrm{d}t \tag{B.14}$$

$$x = \frac{1}{2} \cdot a \cdot t^2 \tag{B.15}$$

The brake distance from (5.1) can be computed by solving (B.13) for t and inserting the result into (B.15):

$$v = a \cdot t \tag{B.16}$$

$$t = \frac{v}{a} \tag{B.17}$$

$$x = \frac{1}{2} \cdot a \cdot \left(\frac{v}{a}\right)^2 \tag{B.18}$$

$$x = \frac{a \cdot v^2}{2 \cdot a^2} \tag{B.19}$$

$$x = \frac{v^2}{2 \cdot a} \tag{B.20}$$

Renaming x to d_{stop} and a to a_{brake} yields:

$$d_{\text{stop}} = \frac{v^2}{2 \cdot a_{\text{brake}}} \quad \square \tag{B.21}$$

The safe velocity v_{safe} (see (5.3)) for a given braking deceleration a_{brake} and a given distance to an obstacle d_{obstacle} can then be very easily computed by solving (B.21) for v (with d_{safe} replaced by d_{obstacle}):

$$d_{\text{obstacle}} = \frac{v_{\text{safe}}^2}{2 \cdot a_{\text{brake}}} \tag{B.22}$$

$$v_{\text{safe}}^2 = d_{\text{obstacle}} \cdot 2 \cdot a_{\text{brake}} \tag{B.23}$$

$$v_{\text{safe}} = \sqrt{2 \cdot d_{\text{obstacle}} \cdot a_{\text{brake}}} \quad \square \tag{B.24}$$

B.3. Calculation of the Value of d_ε for Virtual Obstacle Avoidance

The speed of the robot after accelerating for the duration of one execution cycle is equal to its previous linear velocity $\dot{x}$ plus the velocity gained through the (linear) acceleration: $a_{\max} \cdot t_{\text{cycle}}$. The safe distance can then be calculated by applying (5.1), leading to:

$$d_{\text{safe}} = \frac{(\dot{x} + a_{\max} \cdot t_{\text{cycle}})^2}{2 \cdot a_{\text{brake}}} \tag{B.25}$$

The safe distance d_{safe} is composed of the distance to come to a complete stop d_{stop} (see (5.1)) and the (yet) unknown d_ε. This equation can easily be solved for d_ε.

$$d_{\text{safe}} = d_{\text{stop}} + d_\varepsilon \tag{B.26}$$

$$d_\varepsilon = d_{\text{safe}} - d_{\text{stop}} \tag{B.27}$$

If d_{safe} and d_{stop} are inserted, one can obtain a value for d_ε:

$$d_\varepsilon = d_{\text{safe}} - d_{\text{stop}} \tag{B.28}$$
$$= \frac{(\dot{x} + a_{\max} \cdot t_{\text{cycle}})^2}{2 \cdot a_{\text{brake}}} - \frac{\dot{x}^2}{2 \cdot a_{\text{brake}}} \tag{B.29}$$
$$= \frac{\dot{x}^2 + 2 \cdot \dot{x} \cdot a_{\max} \cdot t_{\text{cycle}} + a_{\max}^2 \cdot t_{\text{cycle}}^2 - \dot{x}^2}{2 \cdot a_{\text{brake}}} \tag{B.30}$$
$$= \frac{2 \cdot \dot{x} \cdot a_{\max} \cdot t_{\text{cycle}} + a_{\max}^2 \cdot t_{\text{cycle}}^2}{2 \cdot a_{\text{brake}}} \quad \square \tag{B.31}$$

B.4. Monotonicity of the Safety Cost Function

The cost function (5.4) is monotonically decreasing with the distance x[1] under the assumptions of $x \geq 0$, $a > 0$ and $b > 0$. To prove this, $f_{a,b}(x)$ can be derived with respect to x for $0 \leq x \leq a$:

$$f_{a,b}(x) = \begin{cases} 1 - (a^{-1}x)^b & 0 \leq x \leq a \\ 0 & x > a \end{cases} \tag{B.32}$$
$$f'_{a,b}(x) = f_{a,b}(x) \frac{\mathrm{d}}{\mathrm{d}x} \tag{B.33}$$
$$= 1 - (a^{-1}x)^b \frac{\mathrm{d}}{\mathrm{d}x} \tag{B.34}$$
$$= -(a^{-1}x)^b \frac{\mathrm{d}}{\mathrm{d}x} \tag{B.35}$$
$$= -(a^{-1})^b \cdot x^b \frac{\mathrm{d}}{\mathrm{d}x} \tag{B.36}$$
$$= -a^{-b} \cdot \left(x^b \frac{\mathrm{d}}{\mathrm{d}x} \right) \tag{B.37}$$
$$= -a^{-b} \cdot b \cdot x^{b-1} \tag{B.38}$$

By looking at the derivative (B.38), it can be seen that under the abovementioned assumptions, the result is always negative for $x > 0$ (and zero at $x = 0$):

$$f'_{a,b}(x) = \underbrace{-\underbrace{a^{-b}}_{>0} \cdot \underbrace{b}_{>0} \cdot \underbrace{x^{b-1}}_{\geq 0}}_{\leq 0} \tag{B.39}$$

[1] Please note that the symbol for the distance has been renamed from d to x, to avoid confusions during differentiation.

Thus, the cost function is monotonically decreasing over the domain $[0, a]$. As the function evaluates to zero beyond values of a, it is also monotonically decreasing[2] over the domain $[0, \infty]$. □

B.5. Integral of the Safety Cost Function

For simplicity, the integration is limited to the domain $0 \leq x \leq a$. For values $x > a$, the cost function (5.4) evaluates to zero and does thus no longer contribute to the integral. For values of $x < 0$ the cost function is undefined. This proof has previously been published in [Arndt 14, 3.3.1].

$$
\begin{aligned}
F_{a,b} &: [0, a] \to \mathbb{R} \\
F_{a,b}(x) &= \int f_{a,b}(x)\,\mathrm{d}x && \text{(B.40)} \\
&= \int 1\,\mathrm{d}x - \int (a^{-1}x)^b\,\mathrm{d}x && \text{(B.41)} \\
&= \int 1\,\mathrm{d}x - \int a^{-b}x^b\,\mathrm{d}x && \text{(B.42)} \\
&= \int 1\,\mathrm{d}x - \frac{1}{a^b}\int x^b\,\mathrm{d}x && \text{(B.43)} \\
&= x - \frac{1}{a^b \cdot (b+1)}x^{b+1} \quad \square && \text{(B.44)}
\end{aligned}
$$

B.6. Incremental Update Rule for the Arithmetic Mean

The rule for calculating the arithmetic mean $\overline{v}_n$ over the first n values v_i can be computed using (B.45). By multiplying both sides with n, (B.46) is obtained.

$$
\overline{v}_n = \frac{1}{n}\sum_i^n v_i \tag{B.45}
$$

$$
\overline{v}_n \cdot n = \sum_i^n v_i \tag{B.46}
$$

If now the value of $\overline{v}_{n+1}$ needs to be computed, $n+1$ can be inserted into (B.45), resulting in (B.47). The last term v_{n+1} can be factored out of the sum according to (B.48). After rewriting and substituting the result from (B.46), the incremental update rule results as given in (B.50).

[2] Although not *strictly* monotonically decreasing, but this was not claimed.

$$\overline{v}_{n+1} = \frac{1}{n+1}\sum_{i}^{n+1} v_i \tag{B.47}$$

$$= \frac{1}{n+1}\left(\sum_{i}^{n} v_i + v_{n+1}\right) \tag{B.48}$$

$$= \frac{\sum_i^n v_i + v_{n+1}}{n+1} \tag{B.49}$$

$$= \frac{\overline{v}_n \cdot n + v_{n+1}}{n+1} \quad \square \tag{B.50}$$

B.7. Cartesian Equation of an Elliptic Cone

An elliptic cone is a cone with an elliptical cross section. The parametric equation of an elliptic cone with its apex at the coordinate $(0, 0, h)^T$ is given by (B.51) to (B.53) [Weisstein 02, p. 880]. The major radius of the elliptical cross section is a, the minor radius b. The height of the cone is h. The parameters of the parametric equation are $v \in [0, 2\pi]$ and $u \in [0, h]$. The base ellipse is visualized in figure B.2

$$x = a \cdot \frac{h-u}{h} \cos v \tag{B.51}$$

$$y = b \cdot \frac{h-u}{h} \sin v \tag{B.52}$$

$$z = u \tag{B.53}$$

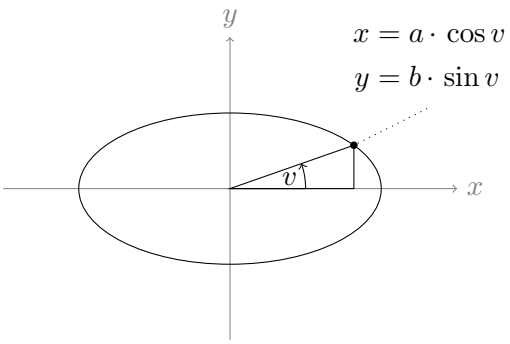


Figure B.2.: Visualization of the parametric equation for the base ellipse of the elliptic cone.

The Cartesian equation for that cone can be calculated by squaring x and y and adding them together:

$$x^2 + y^2 = \left(a\frac{h-u}{h}\right)^2 \cos^2 v + \left(b\frac{h-u}{h}\right)^2 \sin^2 v \tag{B.54}$$

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = \left(\frac{h-u}{h}\right)^2 \cos^2 v + \left(\frac{h-u}{h}\right)^2 \sin^2 v \tag{B.55}$$

$$= \left(\frac{h-u}{h}\right)^2 \underbrace{\cos^2 v + \sin^2 v}_{=1} \tag{B.56}$$

$$= \frac{(h-u)^2}{h^2} \tag{B.57}$$

$$= \frac{(h-z)^2}{h^2} \tag{B.58}$$

Without putting further constraints on the values of x, y and z, this equation describes an infinite elliptic cone with its apex at $(0, 0, h)^T$ (see visualization in figure B.3).

In this work, it is more convenient to have the apex of the cone at $(0, 0, 0)^T$. To achieve this, the value of h can be subtracted from the z value, resulting in:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = \frac{(h-(z-h))^2}{h^2} \tag{B.59}$$

$$= \frac{(-z)^2}{h^2} \tag{B.60}$$

$$= \frac{z^2}{h^2} \tag{B.61}$$

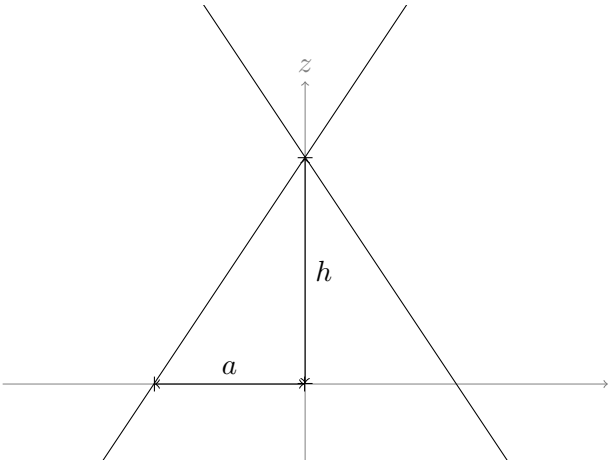


Figure B.3.: Visualization of the infinite elliptic cone with its apex at $(0, 0, h)^T$.

B.8. Intersection of a Line with an Elliptic Cone

When calculating the intersection of a line with an elliptic cone, it will be assumed that the cone has its apex at $(0,0,0)^T$ (see (B.58)). The parametric equation of a line is given by its position vector $(x_0, y_0, z_0)^T$, its direction vector $(r_x, r_y, r_z)^T$ and the parameter t:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} + t \begin{pmatrix} r_x \\ r_y \\ r_z \end{pmatrix} \tag{B.62}$$

Inserting (B.62) into (B.58) yields:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = \frac{z^2}{h^2} \tag{B.63}$$

$$\frac{(x_0 + tr_x)^2}{a^2} + \frac{(y_0 + tr_y)^2}{b^2} = \frac{(z_0 + tr_z)^2}{h^2} \tag{B.64}$$

$$\left(\frac{(x_0 + tr_x)^2}{a^2} + \frac{(y_0 + tr_y)^2}{b^2} \right) h^2 = (z_0 + tr_z)^2 \tag{B.65}$$

$$\left(\frac{x_0^2 + 2tx_0r_x + t^2r_x^2}{a^2} + \frac{y_0^2 + 2ty_0r_y + t^2r_y^2}{b^2} \right) h^2 = z_0^2 + 2tz_0r_z + t^2r_z^2 \tag{B.66}$$

$$\frac{h^2}{a^2}(x_0^2 + 2tx_0r_x + t^2r_x^2) + \frac{h^2}{b^2}(y_0^2 + 2ty_0r_y + t^2r_y^2) - z_0^2 - 2tz_0r_z - t^2r_z^2 = 0 \tag{B.67}$$

$$t^2 \underbrace{\left(\frac{r_x^2h^2}{a^2} + \frac{r_y^2h^2}{b^2} - r_z^2 \right)}_{q_a} + t \underbrace{\left(\frac{2x_0r_xh^2}{a^2} + \frac{2y_0r_yh^2}{b^2} - 2z_0r_z \right)}_{q_b} + \underbrace{\left(\frac{h^2x_0^2}{a^2} + \frac{h^2y_0^2}{b^2} - z_0^2 \right)}_{q_c} = 0 \tag{B.68}$$

This is a quadratic equation which can be solved using the standard quadratic formula:

$$t_{1,2} = \frac{-q_b \pm \sqrt{q_b^2 - 4q_aq_c}}{2q_a} \tag{B.69}$$

The values of t can then be inserted into the original equation of the line (B.62) to obtain the points of intersection.

B.9. Intersection of a Line with a Sphere

In this case, it will also be assumed that the sphere is centered at $(0,0,0)^T$ to facilitate the calculations. The equation of such a sphere with radius r is given as:

$$\boldsymbol{x}^2 = r^2 \tag{B.70}$$

Using the scalar product, this can be rewritten as:

$$x^2 + y^2 + z^2 = r^2 \tag{B.71}$$

The parametric equation of a line is given by its position vector $(x_0, y_0, z_0)^T$, its direction vector $(r_x, r_y, r_z)^T$ and the parameter t:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} + t \begin{pmatrix} r_x \\ r_y \\ r_z \end{pmatrix} \tag{B.72}$$

Inserting (B.72) into (B.71) yields:

$$(x_0 + tr_x)^2 + (y_0 + tr_y)^2 + (z_0 + tr_z)^2 = r^2 \tag{B.73}$$

$$x_0^2 + 2tx_0r_x + t^2r_x^2 + y_0^2 + 2ty_0r_y + t^2r_y^2 + z_0^2 + 2tz_0r_z + t^2r_z^2 = r^2 \tag{B.74}$$

$$t^2 \underbrace{(r_x^2 + r_y^2 + r_z^2)}_{q_a} + t \underbrace{(2x_0r_x + 2y_0r_y + 2z_0r_z)}_{q_b} + \underbrace{x_0^2 + y_0^2 + z_0^2 - r^2}_{q_c} = 0 \tag{B.75}$$

This is a quadratic equation which can be solved using the standard quadratic formula:

$$t_{1,2} = \frac{-q_b \pm \sqrt{q_b^2 - 4q_aq_c}}{2q_a} \tag{B.76}$$

The values of t can then be inserted into the original equation of the line (B.72) to obtain the points of intersection.

B.10. Calculation of Transformation Matrices for the Sensor Model

For the calculation of the transformation matrices for the sensor model, the following matrices will be used in which sin and cos have been abbreviated by s and c, respectively. $R_x(\alpha)$ rotates around the x axis with the roll angle α, $R_y(\beta)$ rotates around the y axis with the pitch angle β, $R_z(\gamma)$ rotates around the z axis with the yaw angle γ. The matrix $T(x, y, z)$ translates along the three axis.

$$R_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & c(\alpha) & -s(\alpha) & 0 \\ 0 & s(\alpha) & c(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{B.77}$$

$$R_y(\beta) = \begin{pmatrix} c(\beta) & 0 & s(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -s(\beta) & 0 & c(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{B.78}$$

$$R_z(\gamma) = \begin{pmatrix} c(\gamma) & -s(\gamma) & 0 & 0 \\ s(\gamma) & c(\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{B.79}$$

$$T(x, y, z) = \begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{B.80}$$

B.10.1. Forward Transformation: From Sensor To Global Coordinate

$$M(\alpha, \beta, \gamma, x, y, z) \tag{B.81}$$

$$= T(x, y, z) R_z(\gamma) R_y(\beta) R_x(\alpha) \tag{B.82}$$

$$= \begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} c(\gamma) & -s(\gamma) & 0 & 0 \\ s(\gamma) & c(\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} R_y(\beta) R_x(\alpha) \tag{B.83}$$

$$= \begin{pmatrix} c(\gamma) & -s(\gamma) & 0 & x \\ s(\gamma) & c(\gamma) & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix} R_y(\beta) R_z(\gamma) \tag{B.84}$$

$$= \begin{pmatrix} c(\gamma) & -s(\gamma) & 0 & x \\ s(\gamma) & c(\gamma) & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} c(\beta) & 0 & s(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -s(\beta) & 0 & c(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} R_z(\gamma) \tag{B.85}$$

$$= \begin{pmatrix} c(\beta)c(\gamma) & -s(\gamma) & s(\beta)c(\gamma) & x \\ c(\beta)s(\gamma) & c(\gamma) & s(\beta)s(\gamma) & y \\ -s(\beta) & 0 & c(\beta) & z \\ 0 & 0 & 0 & 1 \end{pmatrix} R_z(\gamma) \tag{B.86}$$

$$= \begin{pmatrix} c(\beta)c(\gamma) & -s(\gamma) & s(\beta)c(\gamma) & x \\ c(\beta)s(\gamma) & c(\gamma) & s(\beta)s(\gamma) & y \\ -s(\beta) & 0 & c(\beta) & z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & c(\alpha) & -s(\alpha) & 0 \\ 0 & s(\alpha) & c(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{B.87}$$

$$= \begin{pmatrix} c(\beta)c(\gamma) & -c(\alpha)s(\gamma) + s(\alpha)s(\beta)c(\gamma) & s(\alpha)s(\gamma) + c(\alpha)s(\beta)c(\gamma) & x \\ c(\beta)s(\gamma) & c(\alpha)c(\gamma) + s(\alpha)s(\beta)s(\gamma) & -s(\alpha)c(\gamma) + c(\alpha)s(\beta)s(\gamma) & y \\ -s(\beta) & s(\alpha)c(\beta) & c(\alpha)c(\beta) & z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{B.88}$$

B.10.2. Inverse Transformation: From Global to Sensor Coordinate

The inverse transformation works by applying the same operations as in the forward transformation, but with the opposite sign of the arguments and in the opposite order.

Note that due to the symmetries of the sine and cosine function, the following hold:

$$R_x(-\alpha) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & c(-\alpha) & -s(-\alpha) & 0 \\ 0 & s(-\alpha) & c(-\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{B.89}$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & c(\alpha) & s(\alpha) & 0 \\ 0 & -s(\alpha) & c(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{B.90}$$

$$R_y(-\beta) = \begin{pmatrix} c(-\beta) & 0 & s(-\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -s(-\beta) & 0 & c(-\beta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{B.91}$$

$$\begin{pmatrix} c(\beta) & 0 & -s(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ s(\beta) & 0 & c(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{B.92}$$

$$R_z(-\gamma) = \begin{pmatrix} c(-\gamma) & -s(-\gamma) & 0 & 0 \\ s(-\gamma) & c(-\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{B.93}$$

$$\begin{pmatrix} c(\gamma) & s(\gamma) & 0 & 0 \\ -s(\gamma) & c(\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{B.94}$$

Using this additional information, transformation matrix can be calculated:

$$M'(\alpha, \beta, \gamma, x, z, y) \tag{B.95}$$

$$= R_x(-\alpha)R_y(-\beta)R_z(-\gamma)T(-x, -y, -z) \tag{B.96}$$

$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & c(\alpha) & s(\alpha) & 0 \\ 0 & -s(\alpha) & c(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} c(\beta) & 0 & -s(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ s(\beta) & 0 & c(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} R_z(-\gamma)T(-x, -y, -z) \tag{B.97}$$

$$= \begin{pmatrix} c(\beta) & 0 & -s(\beta) & 0 \\ s(\alpha)s(\beta) & c(\alpha) & s(\alpha)c(\beta) & 0 \\ c(\alpha)s(\beta) & -s(\alpha) & c(\alpha)c(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} R_z(-\gamma)T(-x, -y, -z) \tag{B.98}$$

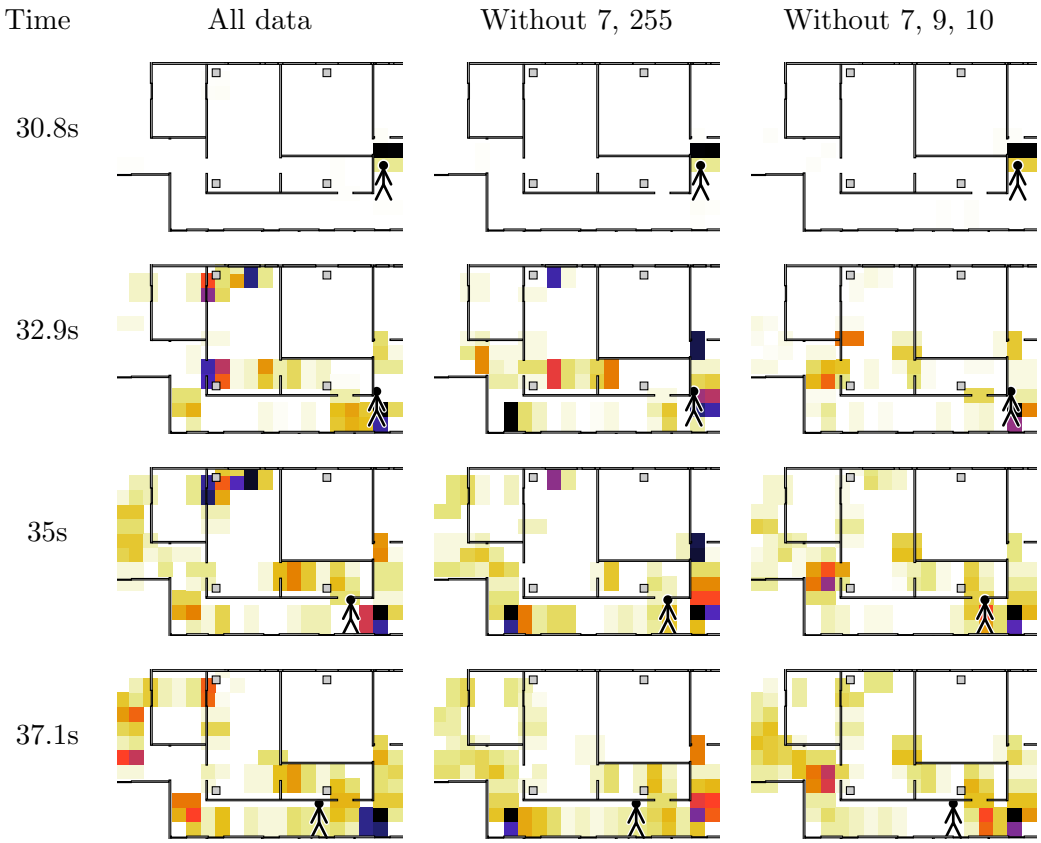
$$= \begin{pmatrix} c(\beta)c(\gamma) & c(\beta)s(\gamma) & -s(\beta) & 0 \\ \begin{matrix} s(\alpha)s(\beta)c(\gamma) \\ -c(\alpha)s(\gamma) \end{matrix} & \begin{matrix} s(\alpha)s(\beta)s(\gamma) \\ +c(\alpha)c(\gamma) \end{matrix} & s(\alpha)c(\beta) & 0 \\ \begin{matrix} c(\alpha)s(\beta)c(\gamma) \\ +s(\alpha)s(\gamma) \end{matrix} & \begin{matrix} c(\alpha)s(\beta)s(\gamma) \\ -s(\alpha)c(\gamma) \end{matrix} & c(\alpha)c(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} T(-x, -y, -z) \tag{B.99}$$

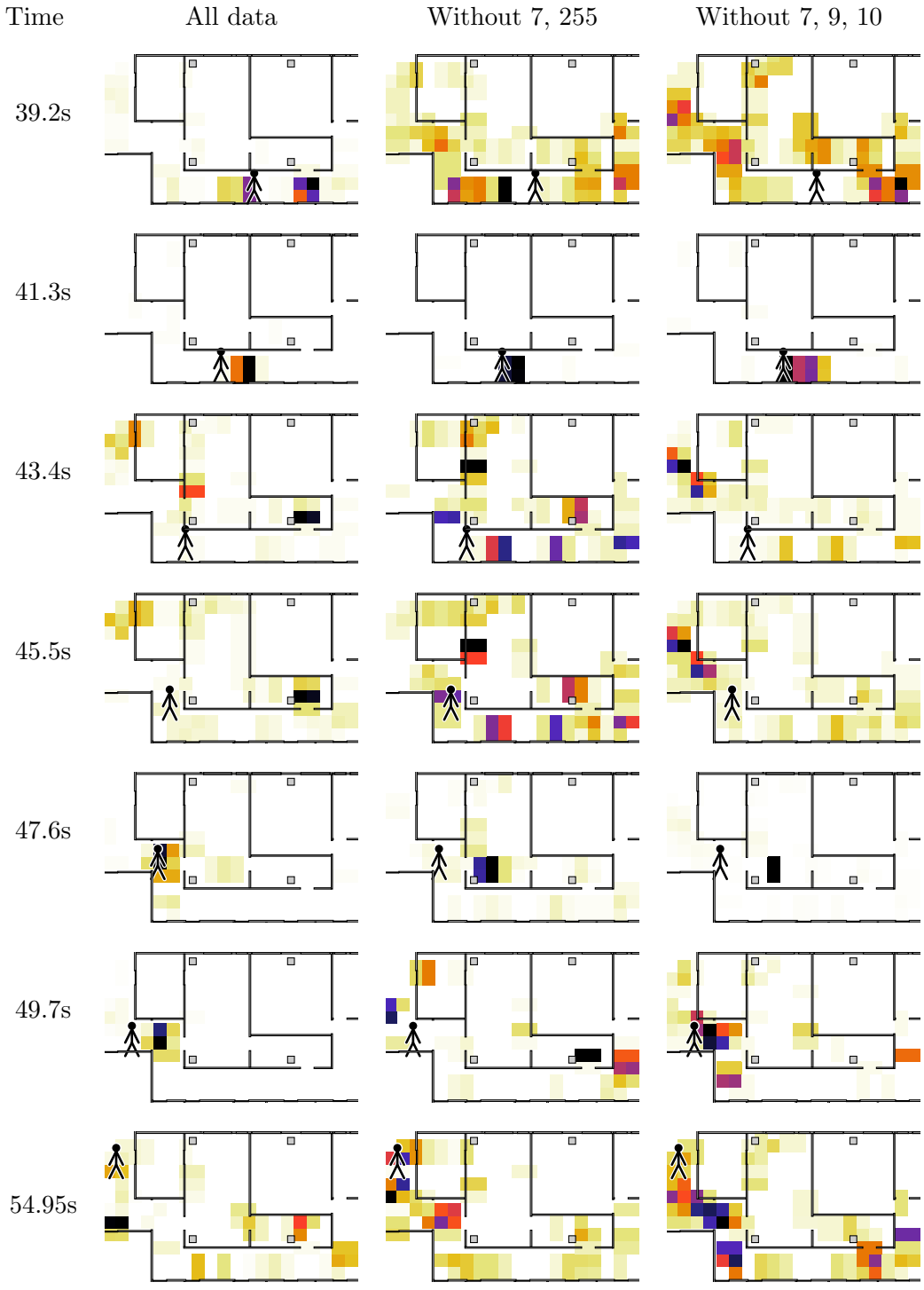
$$= \begin{pmatrix} \underbrace{c(\beta)c(\gamma)}_{R'_{1,1}} & \underbrace{c(\beta)s(\gamma)}_{R'_{1,2}} & \underbrace{-s(\beta)}_{R'_{1,3}} & -xR'_{1,1} - yR'_{1,2} - zR'_{1,3} \\ \underbrace{\begin{matrix} s(\alpha)s(\beta)c(\gamma) \\ -c(\alpha)s(\gamma) \end{matrix}}_{R'_{2,1}} & \underbrace{\begin{matrix} s(\alpha)s(\beta)s(\gamma) \\ +c(\alpha)c(\gamma) \end{matrix}}_{R'_{2,2}} & \underbrace{s(\alpha)c(\beta)}_{R'_{2,3}} & -xR'_{2,1} - yR'_{2,2} - zR'_{2,3} \\ \underbrace{\begin{matrix} c(\alpha)s(\beta)c(\gamma) \\ +s(\alpha)s(\gamma) \end{matrix}}_{R'_{3,1}} & \underbrace{\begin{matrix} c(\alpha)s(\beta)s(\gamma) \\ -s(\alpha)c(\gamma) \end{matrix}}_{R'_{3,2}} & \underbrace{c(\alpha)c(\beta)}_{R'_{3,3}} & -xR'_{3,1} - yR'_{3,2} - zR'_{3,3} \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{B.100}$$

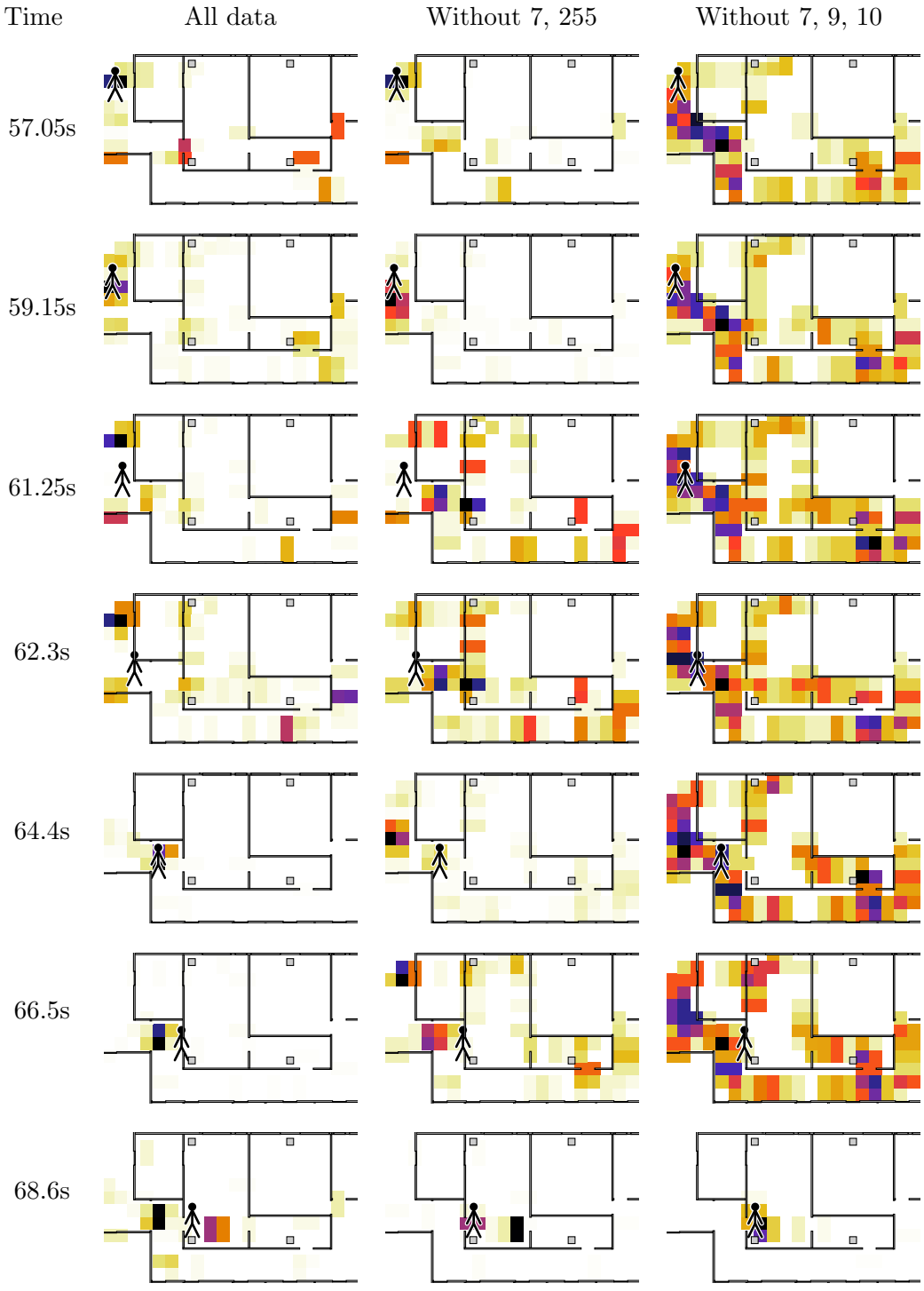
C. Additional Visualizations

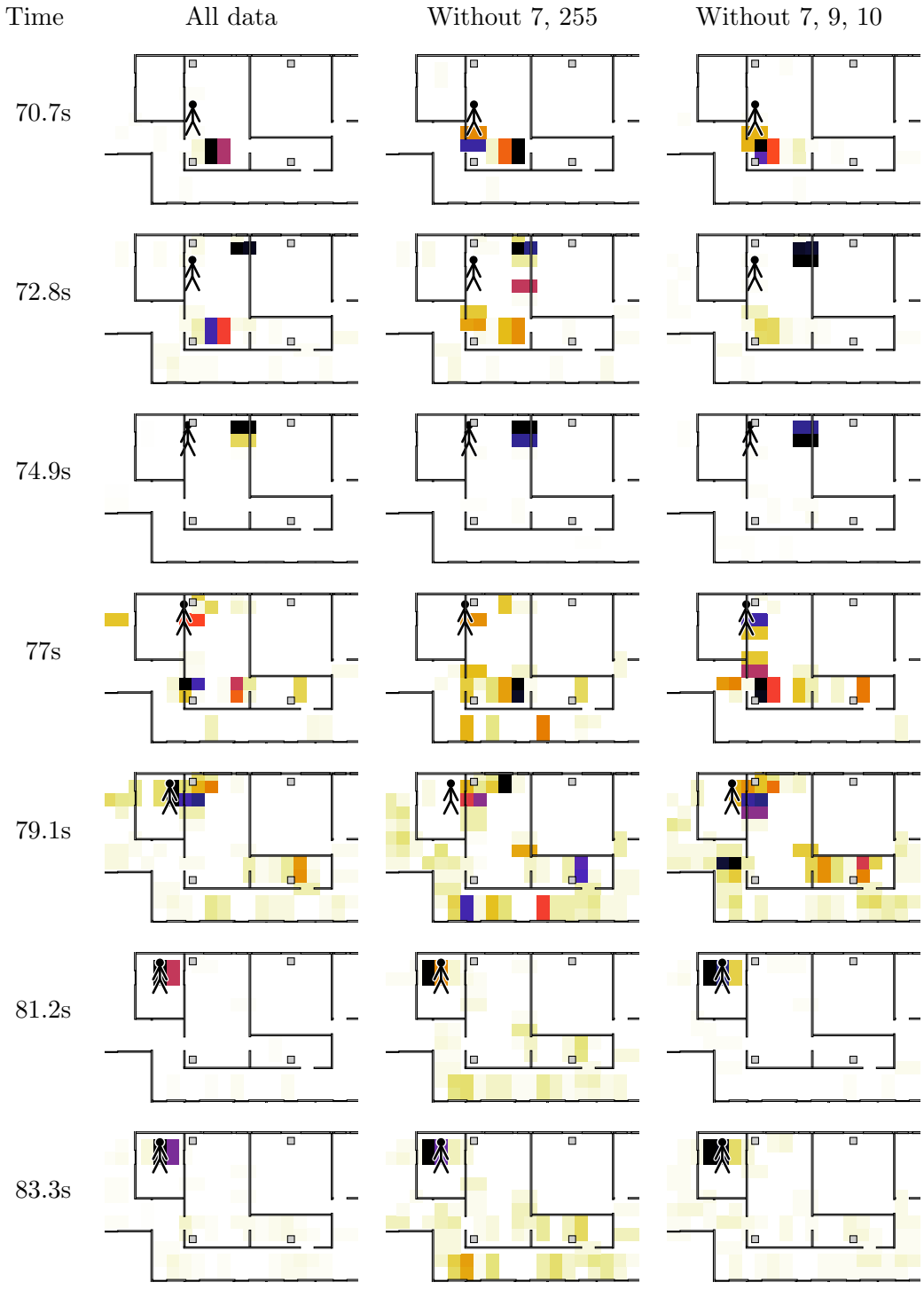
C.1. Single-Target Tracking

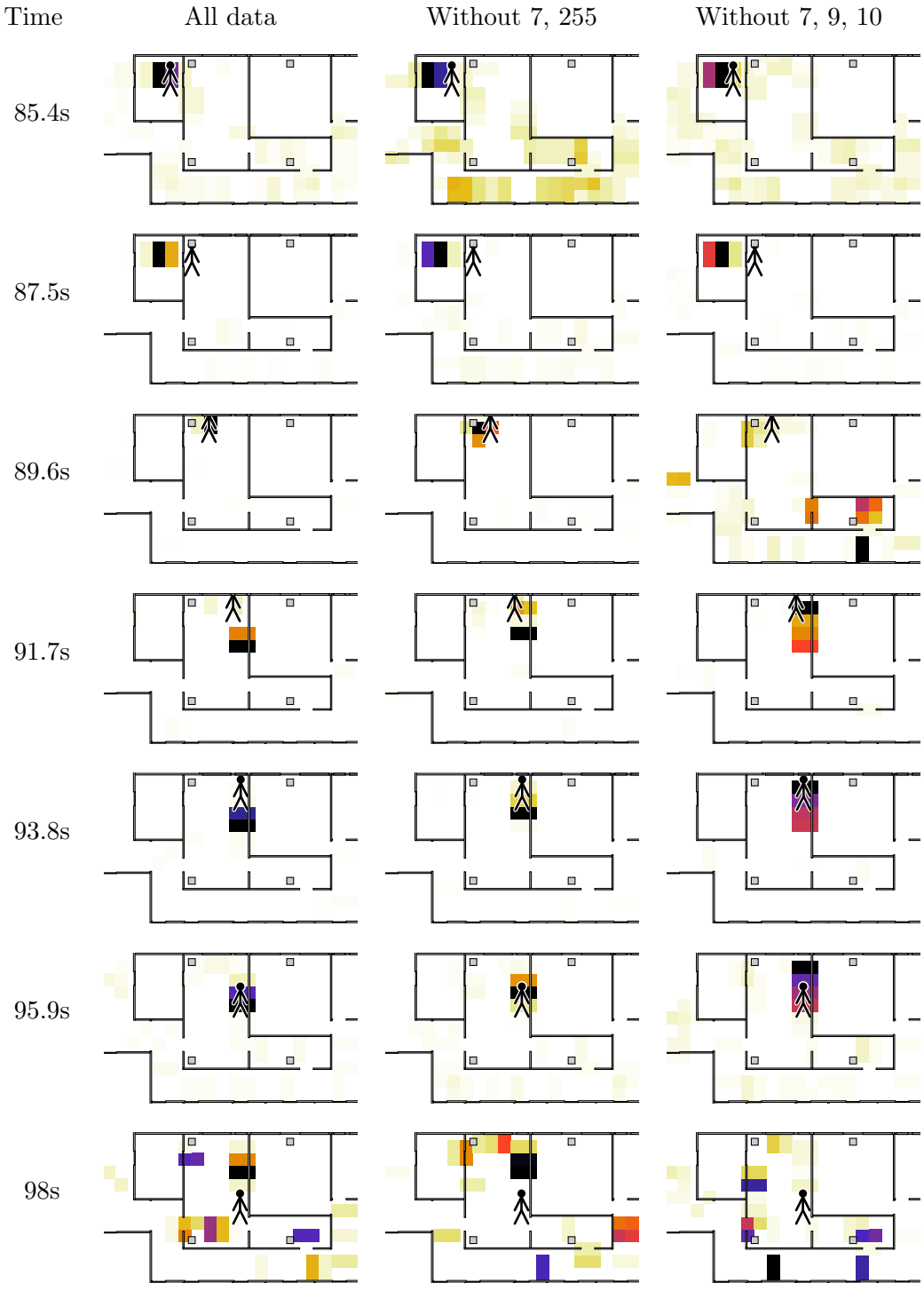
This section shows additional visualizations of the single target tracker as described in section 4.6. Each row shows the same situation, as estimated by a tracker with different sensor configurations. In the first column, all sensors (except the one with identifier 1, which failed the whole time during data acquisition) were active. The second column had nodes 7 and 255 deactivated. In the third column, additionally the nodes with identifiers 9 and 10 were inactive.

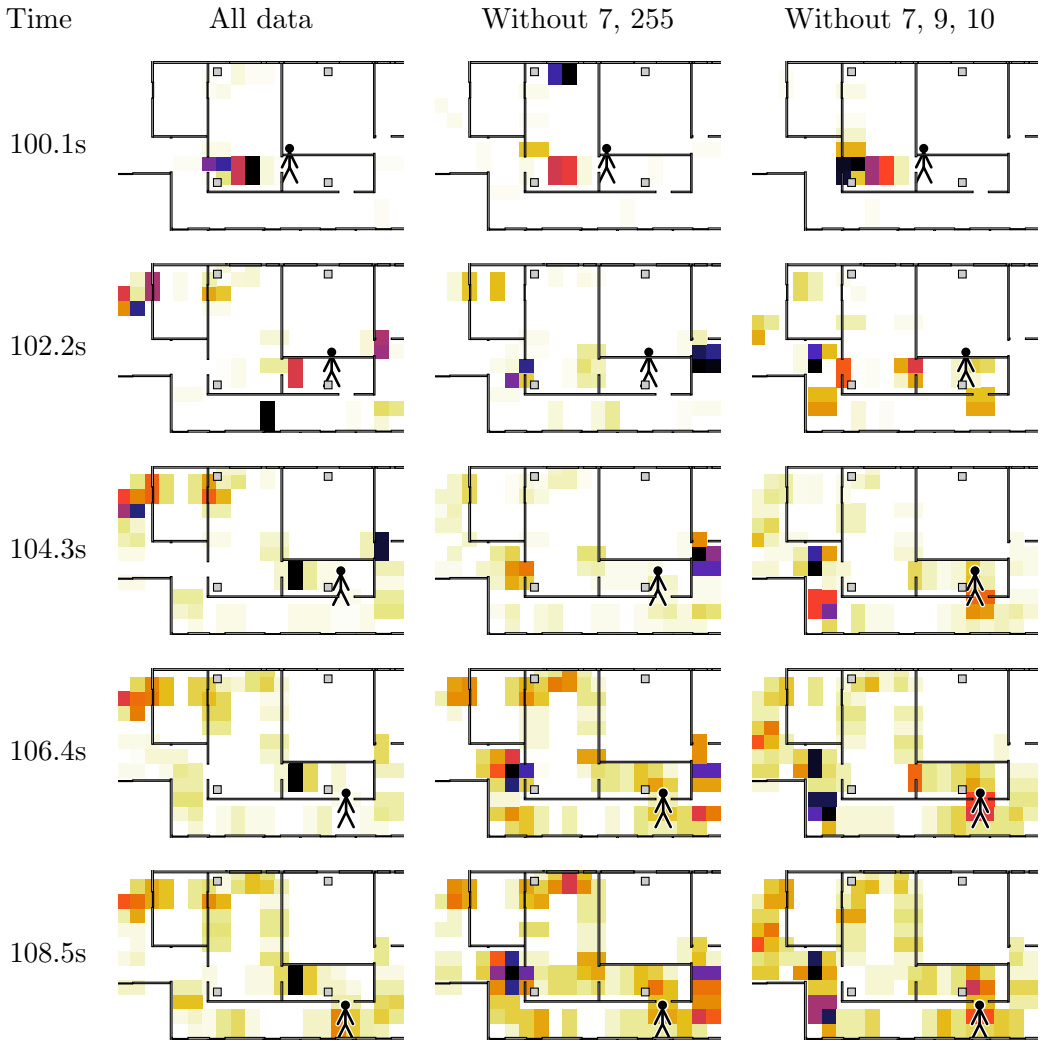



Time
All data
Without 7, 255
Without 7, 9, 10
39.2s
41.3s
43.4s
45.5s
47.6s
49.7s
54.95s


Time
All data
Without 7, 255
Without 7, 9, 10
57.05s
59.15s
61.25s
62.3s
64.4s
66.5s
68.6s

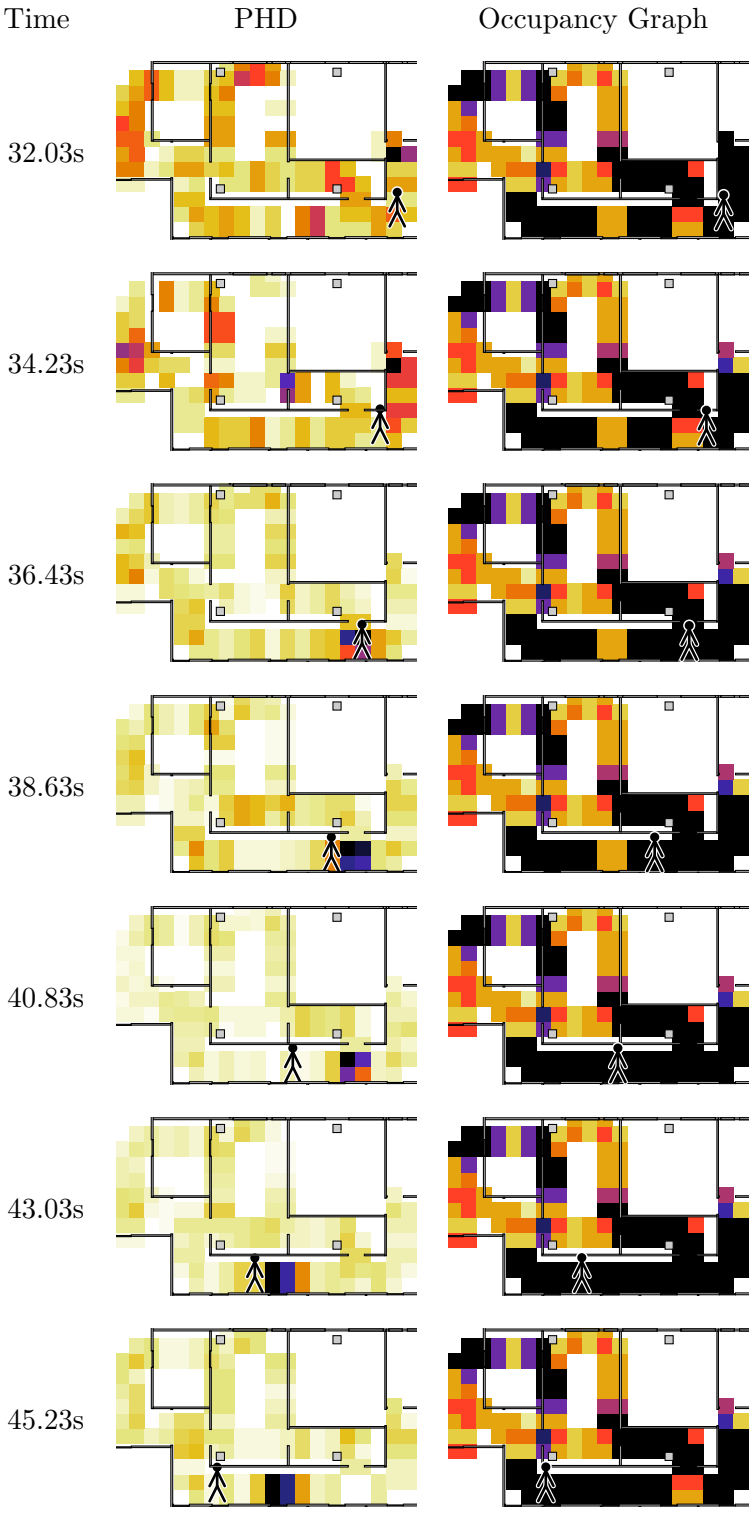

Time
All data
Without 7, 255
Without 7, 9, 10
70.7s
72.8s
74.9s
77s
79.1s
81.2s
83.3s

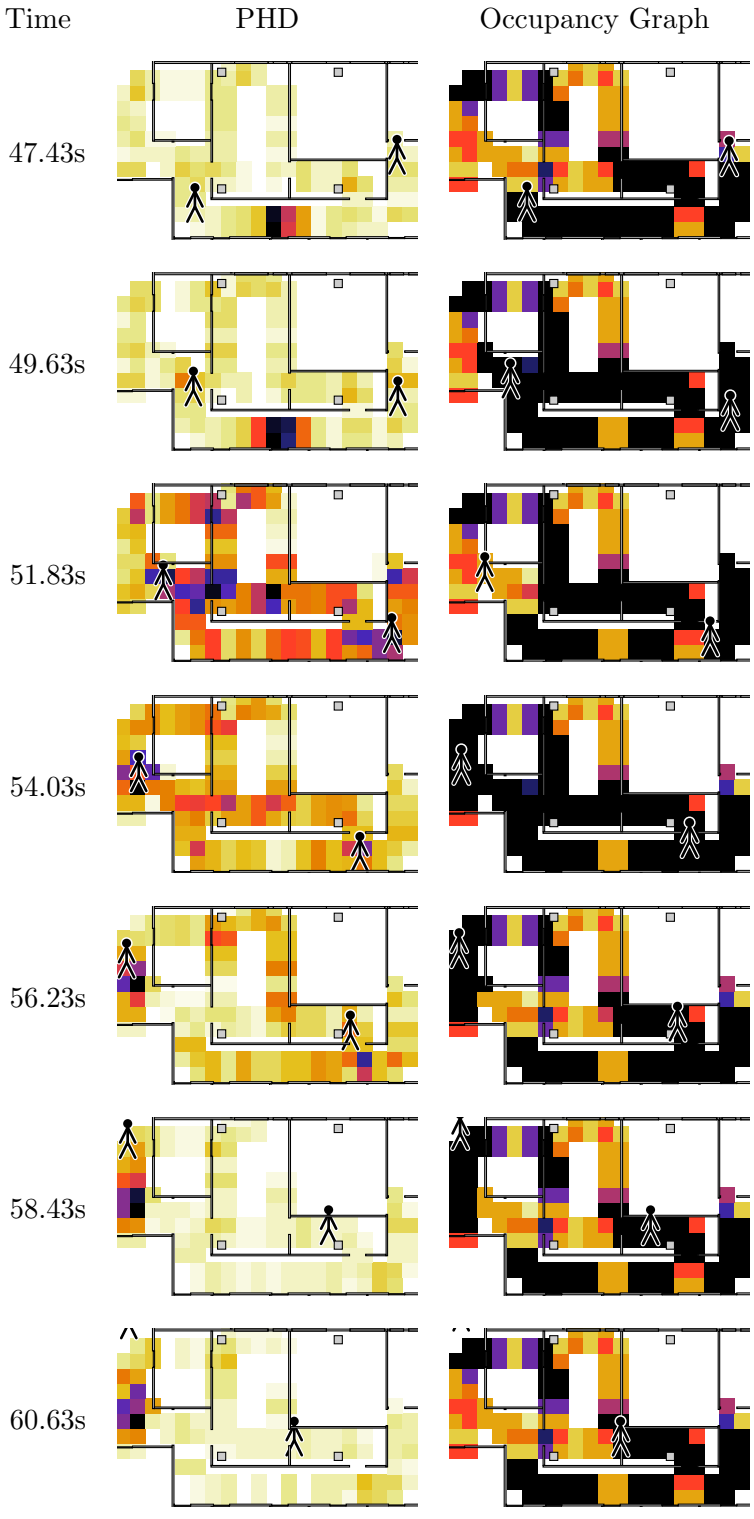

Time
All data
Without 7, 255
Without 7, 9, 10
85.4s
87.5s
89.6s
91.7s
93.8s
95.9s
98s

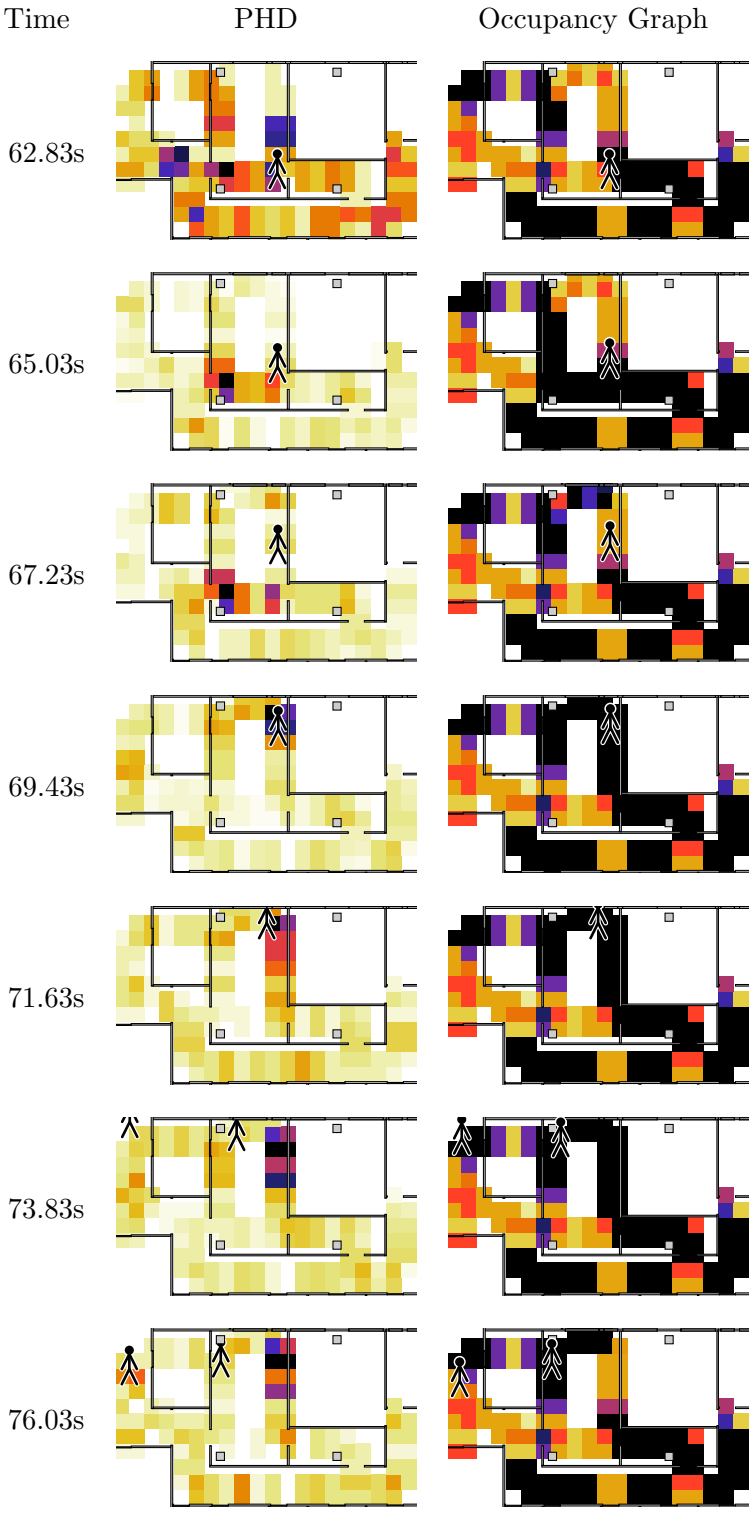


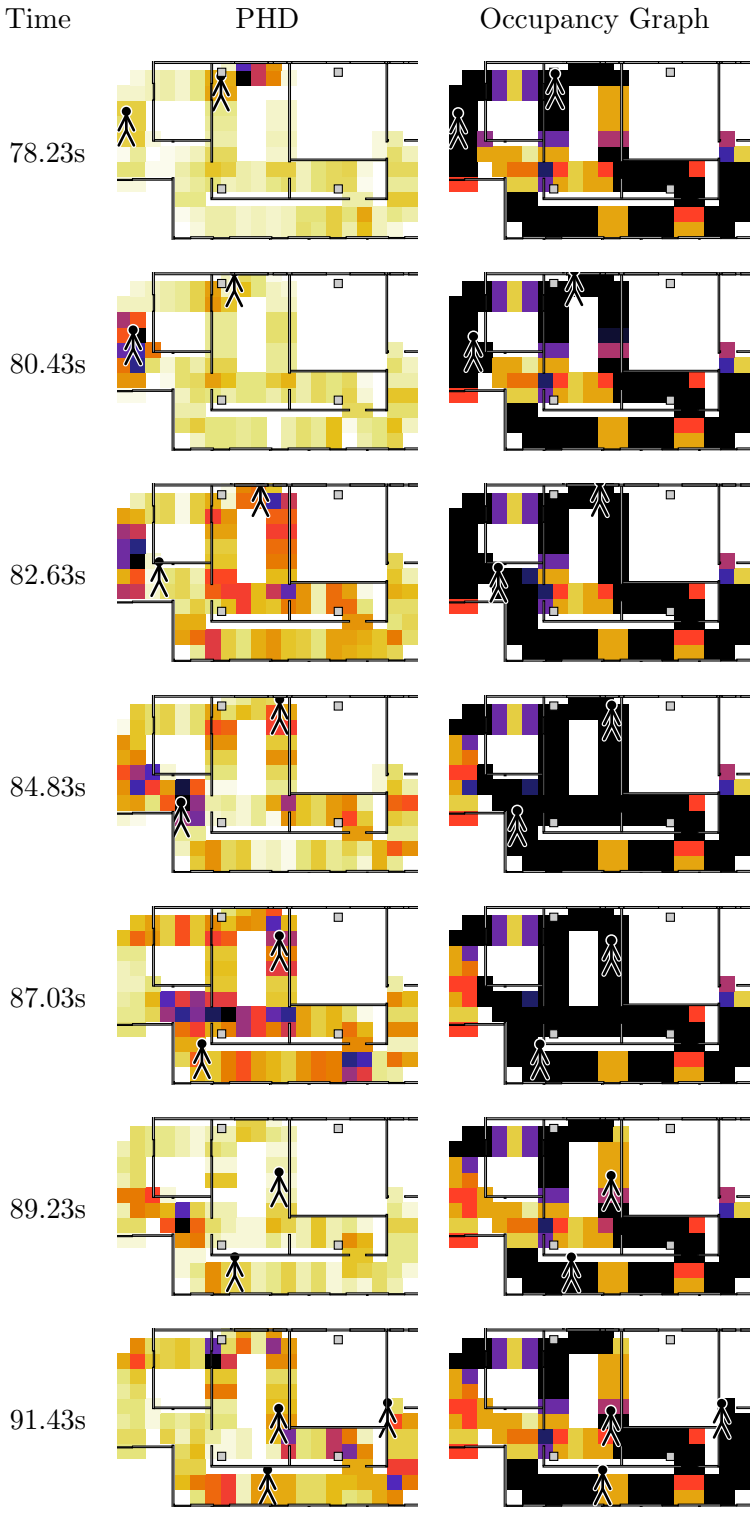
C.2. Multi-Target Tracking and Occupancy Graph Mapping

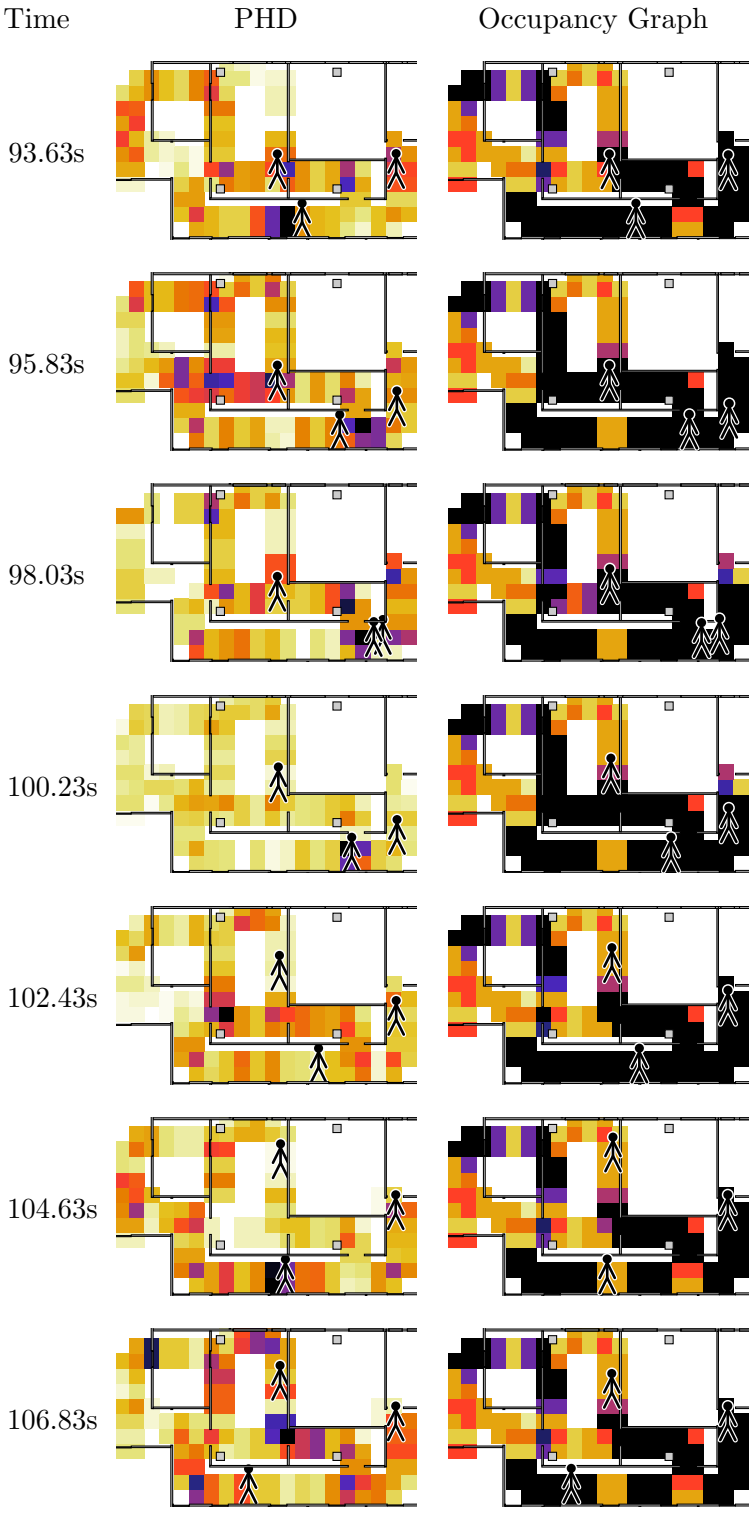
The sequences in this section show the output of the probability hypothesis density (PHD) filter (as described in section 4.7.4) in the left column and the estimation output of the occupancy graph mapping (as described in section 4.8) in the right column. With the help of these sequences, it can be easily seen that the output of the tracker is optimistic, when compared to the (pessimistic) results of the occupancy graph mapping.

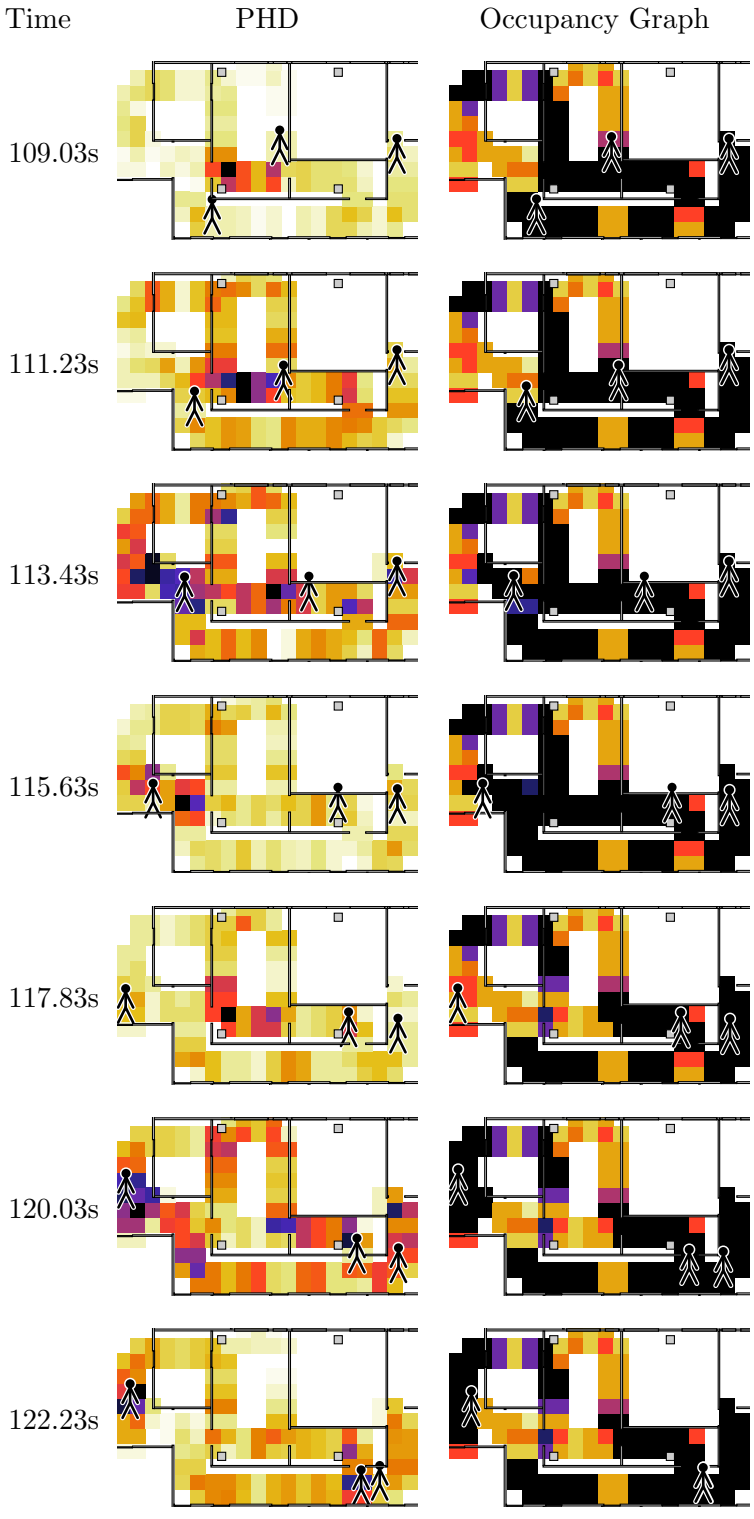

Time
PHD
Occupancy Graph
32.03s
34.23s
36.43s
38.63s
40.83s
43.03s
45.23s

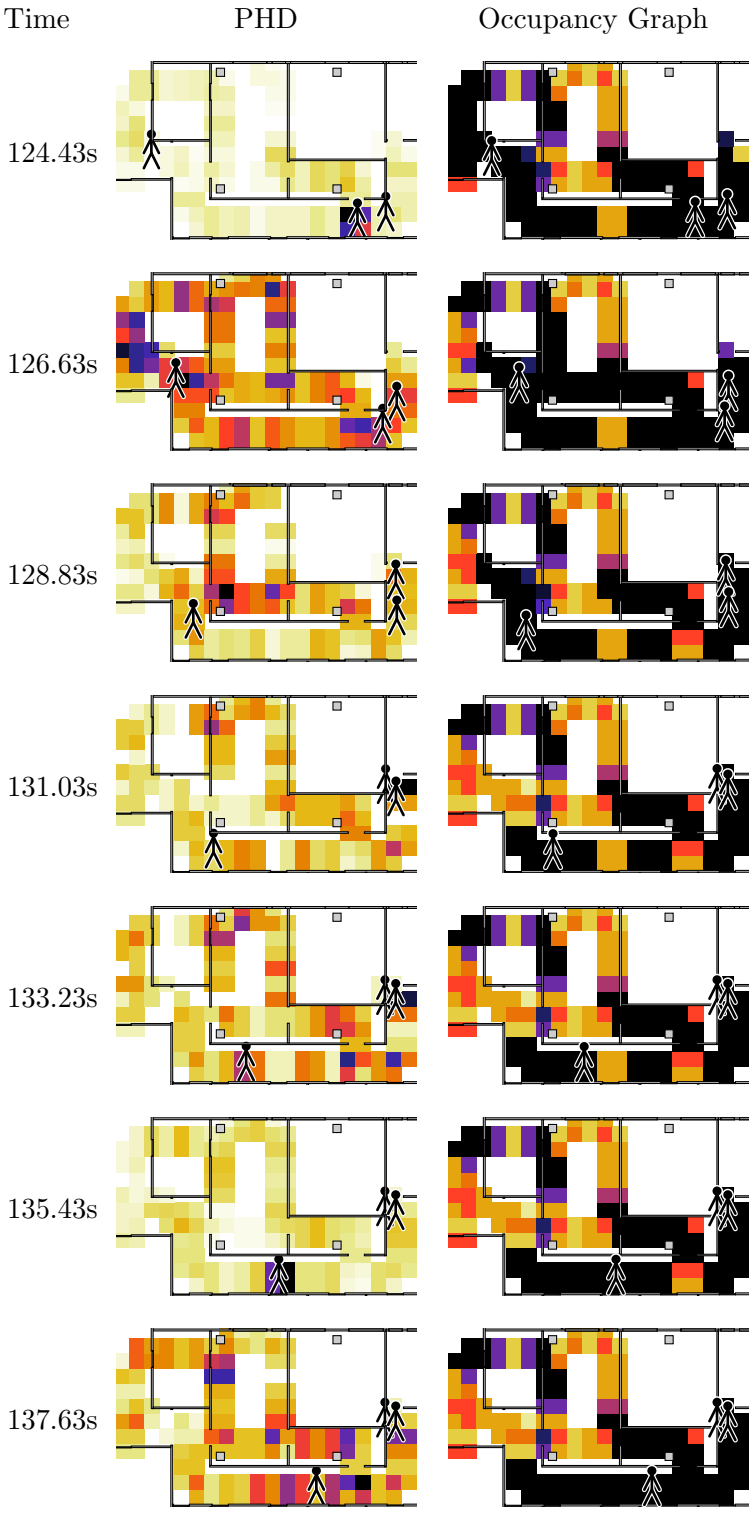

Time
PHD
Occupancy Graph
47.43s
49.63s
51.83s
54.03s
56.23s
58.43s
60.63s

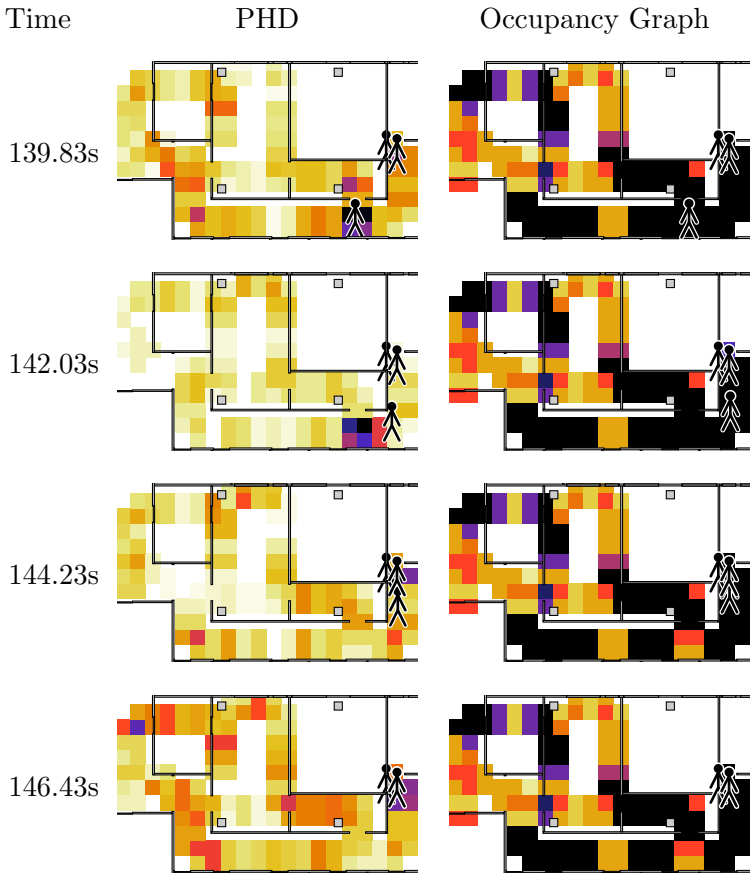

Time
PHD
Occupancy Graph
62.83s
65.03s
67.23s
69.43s
71.63s
73.83s
76.03s


Time
PHD
Occupancy Graph
78.23s
80.43s
82.63s
84.83s
87.03s
89.23s
91.43s


Time
PHD
Occupancy Graph
93.63s
95.83s
98.03s
100.23s
102.43s
104.63s
106.83s

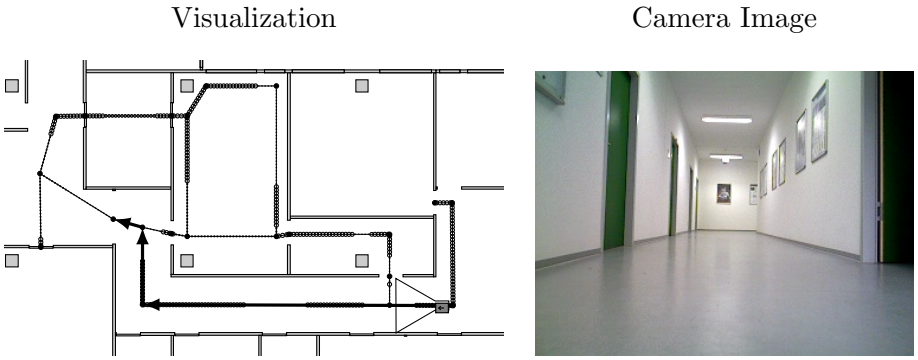

Time
PHD
Occupancy Graph
109.03s
111.23s
113.43s
115.63s
117.83s
120.03s
122.23s


Time
PHD
Occupancy Graph
124.43s
126.63s
128.83s
131.03s
133.23s
135.43s
137.63s

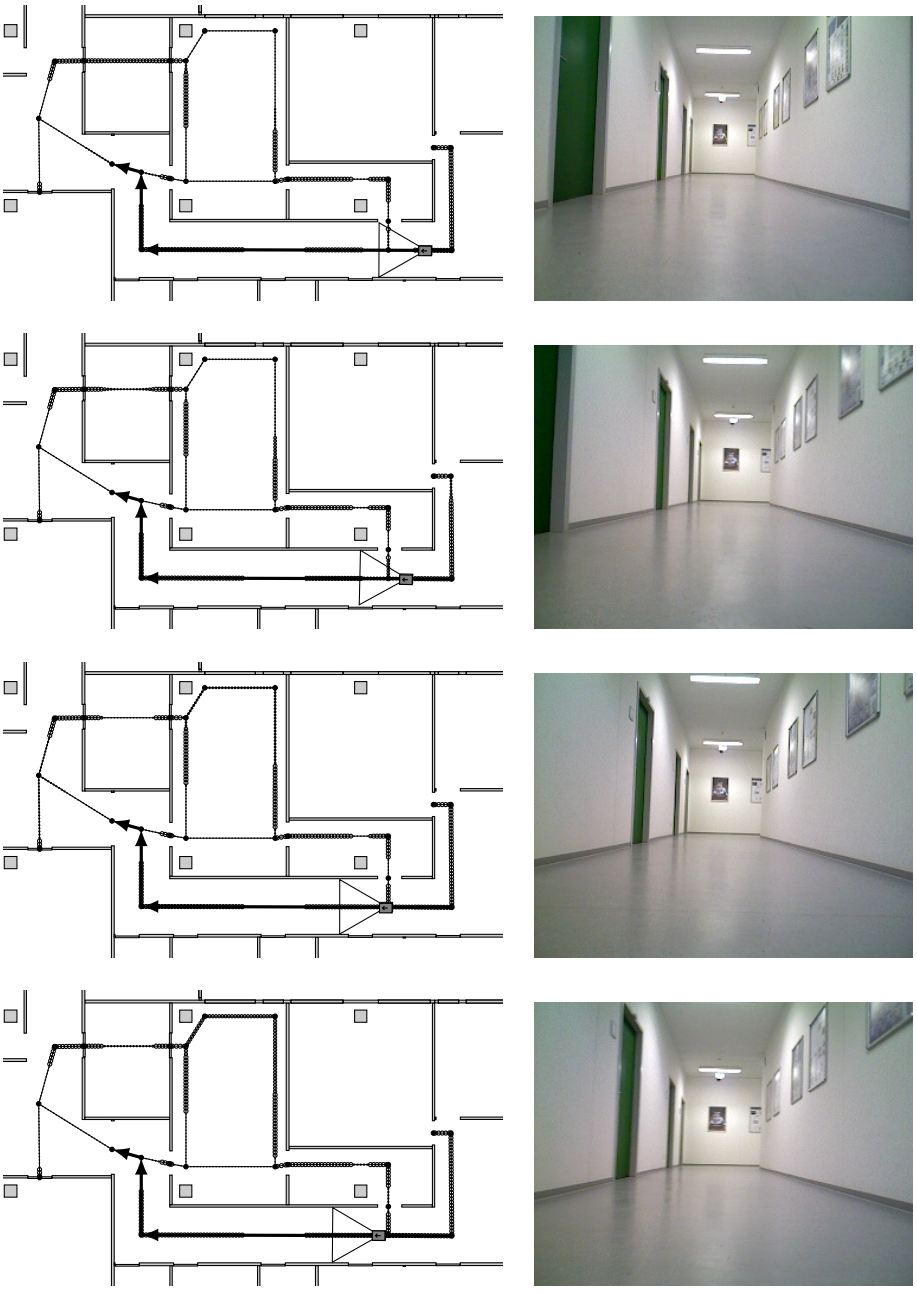


C.3. Virtual Obstacle-Avoidance Sensor

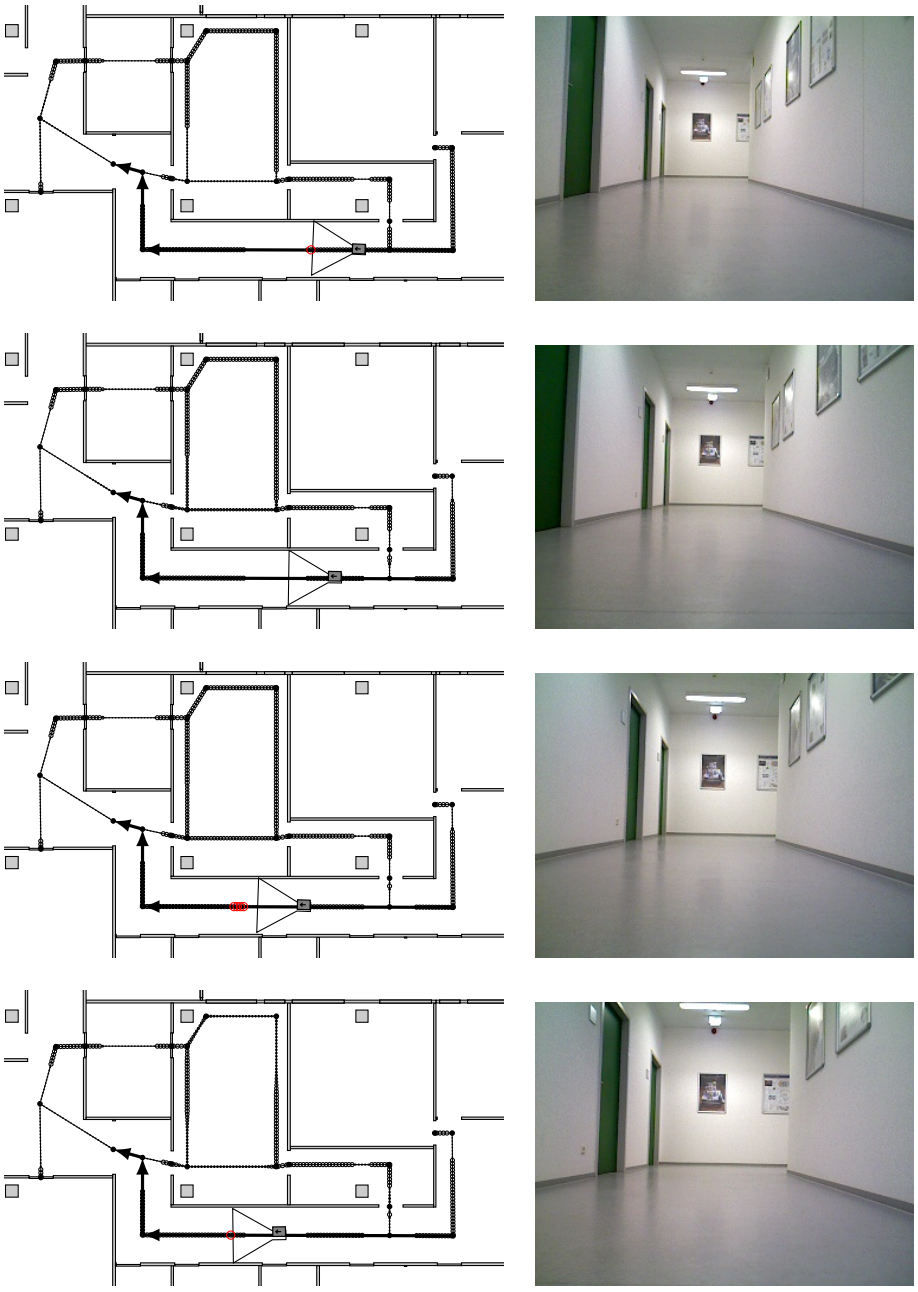
This visualization shows an experiment run to evaluate the virtual obstacle avoidance sensor as described in section 5.1. During that run, all sensor nodes were active and the robot was instructed to drive from place B to A in the environment.



Visualization | Camera Image

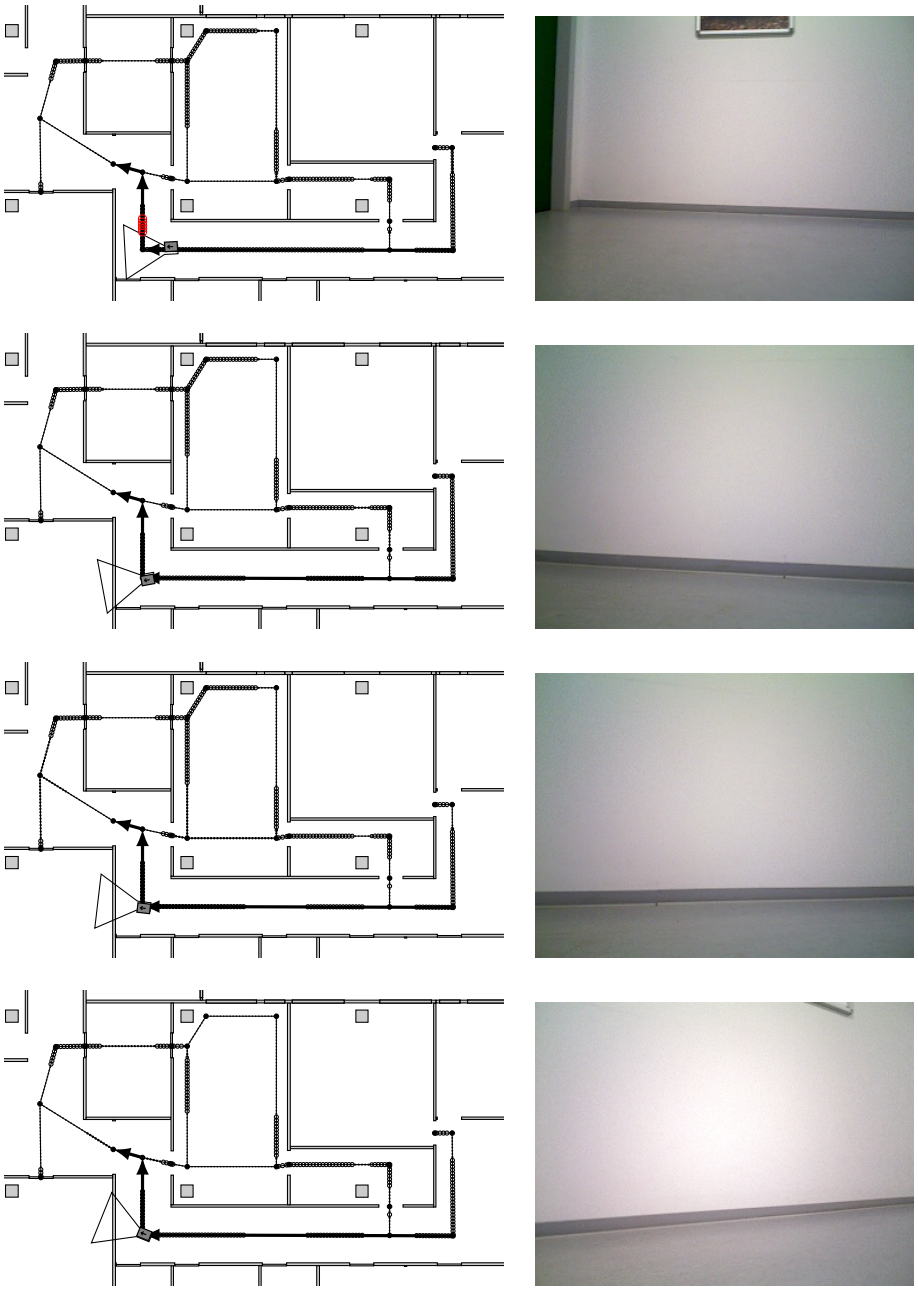


Visualization | Camera Image

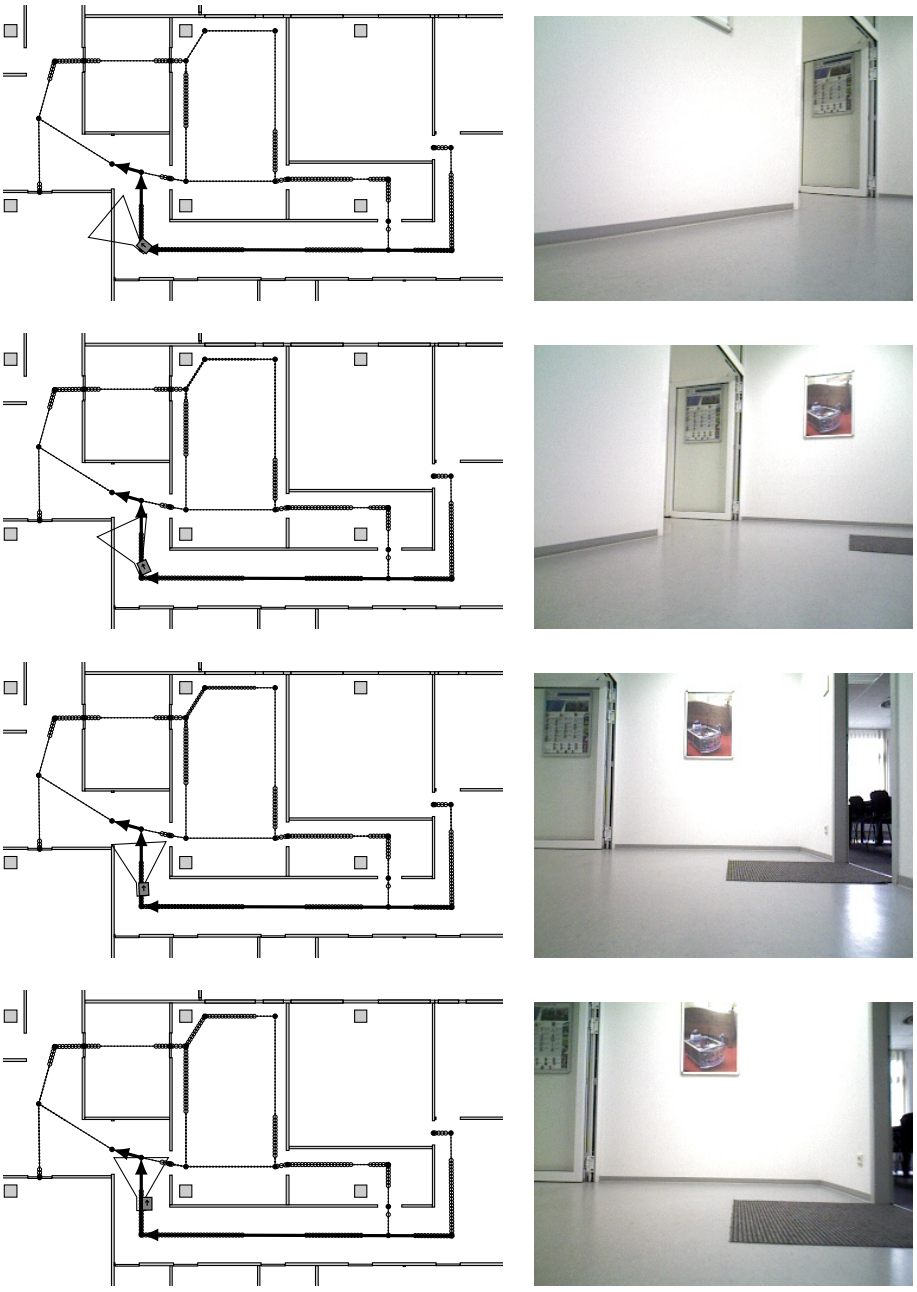


Visualization | Camera Image

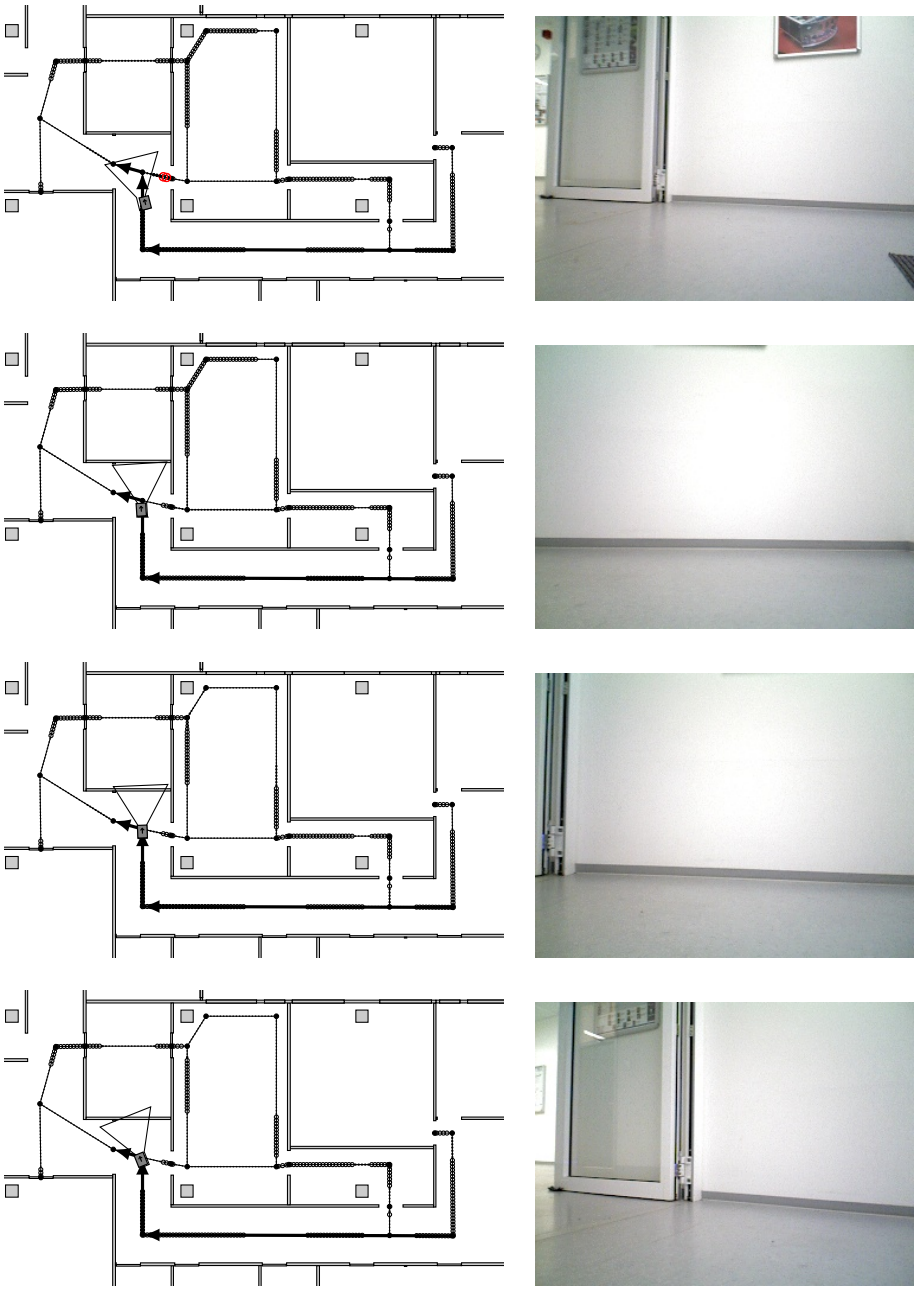
Visualization | Camera Image

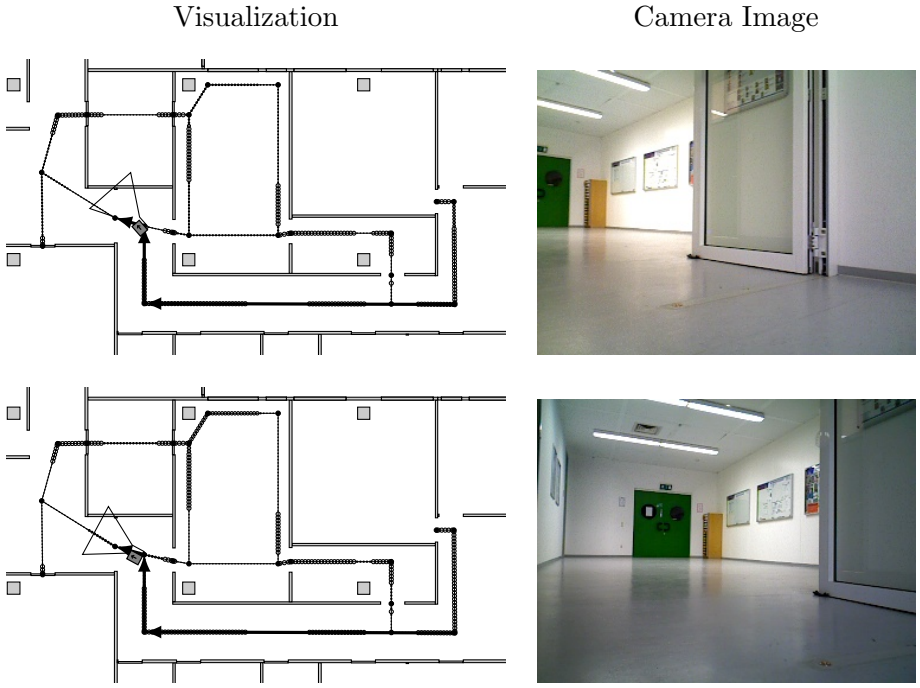


Visualization | Camera Image



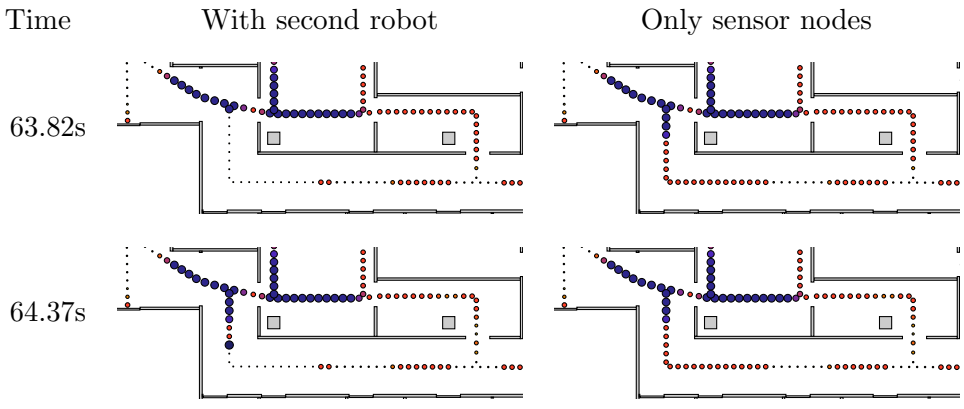
Visualization | Camera Image

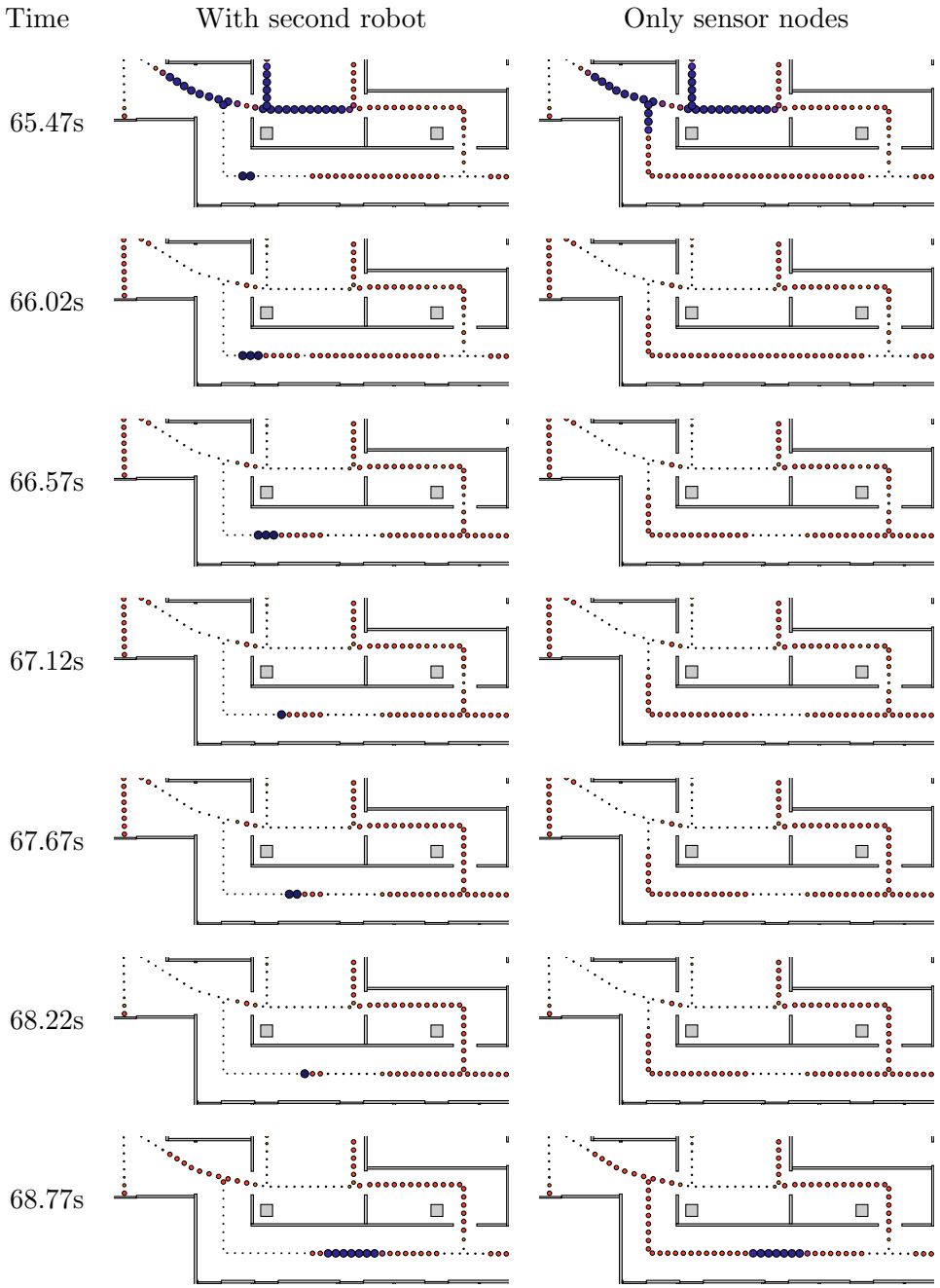




C.4. Enhancement of the State Estimation by a Second Robot

Just as in the main text in section 6.2.2, the left column contains the situations *with* external information from the second robot while the right side does not.




Time
With second robot
Only sensor nodes
65.47s
66.02s
66.57s
67.12s
67.67s
68.22s
68.77s

Bibliography

[Adib 13] F. Adib, Z. Kabelac, D. Katabi, R. C. Miller, "3D Tracking via Body Radio Reflections", in *Usenix NSDI*, vol. 14. 2013. Cited on page 17.

[Alami 06] R. Alami, A. Albu-Schaeffer, A. Bicchi, R. Bischoff, R. Chatila, A. D. Luca, A. D. Santis, G. Giralt, J. Guiochet, G. Hirzinger et al, "Safe and dependable physical human-robot interaction in anthropic domains: State of the art and challenges", in *Proc. IROS*, vol. 6, no. 1. 2006. Cited on page 100.

[Alvarez 04] A. Alvarez, A. Caiti, R. Onken, "Evolutionary Path Planning for Autonomous Underwater Vehicles in a Variable Ocean", *Oceanic Engineering, IEEE Journal of*, vol. 29, no. 2, pp. 418–429, April 2004. Cited on page 112.

[Amato 12a] G. Amato, M. Broxvall, S. Chessa, M. Dragone, C. Gennaro, R. López, L. Maguire, T. M. Mcginnity, A. Micheli, A. Renteria, G. M. P. O'Hare, F. Pecora, "Robotic UBIquitous COgnitive Network", in *Ambient Intelligence - Software and Applications*, ser. Advances in Intelligent and Soft Computing, P. Novais, K. Hallenborg, D. I. Tapia, J. M. C. Rodríguez, Eds., Springer Berlin Heidelberg, 2012, vol. 153, pp. 191–195. Cited on pages 21 and 22.

[Amato 12b] G. Amato, M. Broxvall, S. Chessa, M. Dragone, C. Gennaro, C. Vairo, "When Wireless Sensor Networks Meet Robots", in *ICSNC 2012, The Seventh International Conference on Systems and Networks Communications*. 2012, pp. 35–40. Cited on page 22.

[Amato 15] G. Amato, D. Bacciu, M. Broxvall, S. Chessa, S. Coleman, M. D. Rocco, M. Dragone, C. Gallicchio, C. Gennaro, H. Lozano, T. M. McGinnity, A. Micheli, A. K. Ray, A. Renteria, A. Saffiotti, D. Swords, C. Vairo, P. Vance, "Robotic Ubiquitous Cognitive Ecology for Smart Homes", *Journal of Intelligent & Robotic Systems*, pp. 1–25, 2015. Cited on pages 22, 23, and 27.

[Antonelli 08] G. Antonelli, T. I. Fossen, D. R. Yoerger, "Underwater Robotics", in *Springer Handbook of Robotics*, B. Siciliano, O. Khatib, Eds., Springer Berlin Heidelberg, 2008, ch. 43, pp. 987–1008. Cited on page 7.

[Armbrust 07] C. Armbrust, J. Koch, U. Stocker, K. Berns, "Mobile Robot Navigation Support in Living Environments", in *20. Fachgespräch Autonome Mobile Systeme (AMS)*. Kaiserslautern, Germany: Springer-Verlag, October 2007, pp. 341–346. Cited on page 147.

[Arndt 11a] M. Arndt, "Improving Mobile Robot Navigation by means of Probabilistic Tracking of People within a Wireless Sensor Network", Master's thesis, Robotics Research Lab, Department of Computer Sciences, University of Kaiserslautern,

December 2011. unpublished. Cited on pages 34, 36, 40, 48, 51, 61, 70, 85, 86, 93, and 147.

[Arndt 11b] M. Arndt. "Simulation of Wireless Sensor Networks in the MCA2/finroc Framework". Project Report for the PhD Program (Unpublished), 2011. Cited on pages 159 and 161.

[Arndt 12a] M. Arndt. "Evaluation of the AmICA Wireless Sensor Network Platform for Smart Environments comprising Mobile Robots". Project Report for the PhD Program (Unpublished), March 2 2012. Cited on page 119.

[Arndt 12b] M. Arndt, K. Berns, "Mobile Robots in Smart Environments: The Current Situation", in *Autonomous Mobile Systems 2012*, ser. Informatik aktuell, P. Levi, O. Zweigle, K. Häußermann, B. Eckstein, Eds. Springer Berlin Heidelberg, 2012, pp. 39–47. 10.1007/978-3-642-32217-4_5. Cited on pages 86, 119, 122, 159, and 161.

[Arndt 12c] M. Arndt, K. Berns, "Optimized Mobile Indoor Robot Navigation through probabilistic Tracking of People in a Wireless Sensor Network", in *Proceedings of the 7th German Conference on Robotics (Robotik 2012)*. Munich, Germany: VDI Verlag, Berlin, May 21–22 2012, pp. 355–360. Cited on pages 40, 61, 70, 85, and 93.

[Arndt 13a] M. Arndt, M. Reichardt, J. Hirth, K. Berns, "Requirements for Interoperability and Seamless Integration of Different Robotic Frameworks", in *Proceedings of the eighth full-day Workshop on Software Development and Integration in Robotics (SDIR VIII), in conjunction with the IEEE International Conference on Robotics and Automation (ICRA)*, D. Brugali, Ed. Karlsruhe, Germany, May 2013, pp. 38–40. Cited on page 151.

[Arndt 13b] M. Arndt, S. Wille, L. de Souza, V. F. Rey, N. Wehn, K. Berns, "Performance evaluation of ambient services by combining robotic frameworks and a smart environment platform", *Robotics and Autonomous Systems*, vol. 61, no. 11, pp. 1173 – 1185, 2013. Cited on pages 161 and 162.

[Arndt 14] M. Arndt, K. Berns, "Risk-Reduced Mobile Robot Path Planning in Smart Environments", in *Proceedings for the joint conference of ISR 2014 and ROBOTIK 2014*. VDE VERLAG GMBH, 2014, pp. 582–589. Cited on pages 98, 99, 101, 103, 105, 106, 109, and 168.

[Arndt 15] M. Arndt, K. Berns, "Safe Predictive Mobile Robot Navigation in Aware Environments", in *Proceedings of the 12th International Conference on Informatics in Control, Automation and Robotics (ICINCO 2015)*, O. G. Joaquim Filipe, Kurosh Madani, J. Sasiadek, Eds., vol. 2. Colmar, France, July 21–23 2015, pp. 15–23. Cited on pages 111, 113, 114, and 115.

[Arulampalam 02] M. S. Arulampalam, S. Maskell, N. Gordon, T. Clapp, "A tutorial on Particle Filters for Online Nonlinear/Non-Gaussian Bayesian Tracking", *Signal Processing, IEEE Transactions on*, vol. 50, no. 2, pp. 174–188, February 2002. Cited on pages 32, 33, and 37.

[Asimov 13] I. Asimov, *I, Robot*, Harpercollins Publishers, United Kingdom, 2013. Cited on page 98.

[Augusto 10] J. C. Augusto, H. Nakashima, H. Aghajan, "Ambient Intelligence and Smart Environments: A State of the Art", in *Handbook of Ambient Intelligence and Smart Environments*, Springer, 2010, pp. 3–31. Cited on page 1.

[Bacciu 11] D. Bacciu, C. Gallicchio, A. Micheli, S. Chessa, P. Barsocchi, "Predicting user movements in heterogeneous indoor environments by reservoir computing", in *Proc. of the IJCAI Workshop on Space, Time and Ambient Intelligence (STAMI), Barcellona, Spain*, Citeseer. 2011, pp. 1–6. Cited on pages 22 and 28.

[Bacciu 12] D. Bacciu, M. Broxvall, S. Coleman, M. Dragone, C. Gallicchio, C. Gennaro, R. Guzmán, R. López, H. Lozano-Peiteado, A. Ray, A. Renteria, A. Saffiotti, C. Vairo, "Self-sustaining Learning for Robotic Ecologies", in *Proceedings of the 2012 International Conference on Sensor Networks (SENSORNETS'12*. 2012, pp. 99–103. Cited on page 22.

[Bacciu 14a] D. Bacciu, C. Gallicchio, A. Micheli, M. D. Rocco, A. Saffiotti, "Learning context-aware mobile robot navigation in home environments", in *Information, Intelligence, Systems and Applications, IISA 2014, The 5th International Conference on*. July 2014, pp. 57–62. Cited on page 22.

[Bacciu 14b] D. Bacciu, P. Barsocchi, S. Chessa, C. Gallicchio, A. Micheli, "An experimental characterization of reservoir computing in ambient assisted living applications", *Neural Computing and Applications*, vol. 24, no. 6, pp. 1451–1464, 2014. Cited on page 22.

[Bar-Shalom 78] Y. Bar-Shalom, "Tracking Methods in a Multitarget Environment", *Automatic Control, IEEE Transactions on*, vol. 23, no. 4, pp. 618–626, Aug 1978. Cited on page 34.

[Barsocchi 11] P. Barsocchi, S. Chessa, E. Ferro, F. Furfari, F. Potorti, "Context driven enhancement of RSS-based localization systems", in *Computers and Communications (ISCC), 2011 IEEE Symposium on*. June 2011, pp. 463–468. Cited on pages 22 and 28.

[Behnke 04] S. Behnke, "Local Multiresolution Path Planning", in *RoboCup 2003: Robot Soccer World Cup VII*, ser. Lecture Notes in Computer Science, D. Polani, B. Browning, A. Bonarini, K. Yoshida, Eds., Springer Berlin Heidelberg, 2004, vol. 3020, pp. 332–343. Cited on page 14.

[Bennewitz 05] M. Bennewitz, W. Burgard, G. Cielniak, S. Thrun, "Learning Motion Patterns of People for Compliant Robot Motion", *The International Journal of Robotics Research*, vol. 24, no. 1, pp. 31–48, 2005. Cited on page 39.

[Berns 10] K. Berns, S. A. Mehdi, "Use of an Autonomous Mobile Robot for Elderly Care", in *Advanced Technologies for Enhancing Quality of Life*, ser. AT-EQUAL '10. Iasi, Romania: IEEE Computer Society, July 15–19 2010, pp. 121–126. Cited on page 147.

[Bevilacqua 15] R. Bevilacqua, E. Felici, F. Marcellini, S. Glende, S. Klemcke, I. Conrad, R. Esposito, F. Cavallo, P. Dario, "Robot-Era Project: Preliminary Results on the System Usability", in *Design, User Experience, and Usability: Interactive Experience Design*, ser. Lecture Notes in Computer Science, A. Marcus, Ed., Springer International Publishing, 2015, vol. 9188, pp. 553–561. Cited on pages 23 and 24.

[Blackman 04] S. S. Blackman, "Multiple hypothesis tracking for multiple target tracking", *Aerospace and Electronic Systems Magazine, IEEE*, vol. 19, no. 1, pp. 5–18, Jan 2004. Cited on page 75.

[Blaunstein 07] N. Blaunstein, C. G. Christodoulou, *Radio propagation and adaptive antennas for wireless communication links: terrestrial, atmospheric and ionospheric*, ser. Wiley series in microwave and optical engineering, Wiley-Interscience, 2007. Cited on page 120.

[Boccuzzi 08] J. Boccuzzi, *Signal processing for wireless communications*, McGraw-Hill, 2008. Cited on page 120.

[Bordignon 07] M. Bordignon, J. Rashid, M. Broxvall, A. Saffiotti, "Seamless integration of Robots and Tiny Embedded Devices in a PEIS-Ecology", in *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, IEEE. 2007, pp. 3101–3106. Cited on pages 20 and 21.

[Broxvall 06a] M. Broxvall, S. Coradeschi, A. Loutfi, A. Saffiotti, "An Ecological Approach to Odour Recognition in Intelligent Environments", in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. Orlando, FL, 2006, pp. 2066–2071. Cited on pages 21 and 29.

[Broxvall 06b] M. Broxvall, M. Gritti, A. Saffiotti, B. S. Seo, Y. J. Cho, "PEIS Ecology: Integrating Robots into Smart Environments", in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*. Orlando, FL, 2006, pp. 212–218. Cited on pages 20 and 29.

[Broz 12] F. Broz, A. Nuovo, T. Belpaeme, A. Cangelosi, "Multimodal robot feedback for eldercare", in *Workshop on Robot Feedback in Human-Robot Interaction: How to make a Robot Readable for a Human Interaction Partner at Ro-Man, Ro-Man*, vol. 12. 2012. Cited on page 23.

[Buehler 07] M. Buehler, K. Iagnemma, S. Singh, *The 2005 DARPA Grand Challenge: The Great Robot Race*, 1st ed., Springer Publishing Company, Incorporated, 2007. Cited on page 1.

[Buehler 09] M. Buehler, K. Iagnemma, S. Singh, *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic*, ser. Springer Tracts in Advanced Robotics, Springer Berlin Heidelberg, 2009. Cited on page 1.

[Burgard 08] W. Burgard, M. Hebert, "World Modeling", in *Springer Handbook of Robotics*, B. Siciliano, O. Khatib, Eds., Springer Berlin Heidelberg, 2008, ch. 36, pp. 853–869. Cited on pages 8 and 9.

[Cadman 03] J. Cadman, "Deploying Commercial Location-Aware systems", in *Proceedings of the 2003 Workshop on Location-Aware Computing (held as part of UbiComp 2003)*. 2003, pp. 4–6. Cited on page 11.

[Campion 08] G. Campion, W. Chung, "Wheeled Robots", in *Springer Handbook of Robotics*, B. Siciliano, O. Khatib, Eds., Springer Berlin Heidelberg, 2008, ch. 17, pp. 391–410. Cited on page 7.

[Cavallo 12] F. Cavallo, M. Aquilano, M. C. Carrozza, P. Dario, "Robot-era project: The vision of 3d service robotics", *Gerontechnology*, vol. 11, no. 2, p. 364, 2012. Cited on page 23.

[Cirillo 10] M. Cirillo, L. Karlsson, A. Saffiotti, "Human-aware Task Planning: An Application to Mobile Robots", *ACM Transactions on Intelligent Systems and Technology*, vol. 1, no. 2, pp. 15:1–15:26, December 2010. Cited on page 100.

[Cirillo 12] M. Cirillo, L. Karlsson, A. Saffiotti, "Human-aware planning for robots embedded in ambient ecologies", *Pervasive and Mobile Computing*, vol. 8, no. 4, pp. 542 – 561, 2012. Special Issue on Ambient Ecologies. Cited on page 100.

[Coradeschi 06] S. Coradeschi, A. Saffiotti, "Symbiotic Robotic Systems: Humans, Robots, and Smart Environments", *Intelligent Systems, IEEE*, vol. 21, no. 3, pp. 82–84, 2006. Cited on page 26.

[Crowley 85] J. L. Crowley, "Navigation for an Intelligent Mobile Robot", *Robotics and Automation, IEEE Journal of*, vol. 1, no. 1, pp. 31–41, March 1985. Cited on page 14.

[Dalal 06] N. Dalal, B. Triggs, C. Schmid, "Human Detection Using Oriented Histograms of Flow and Appearance", in *Computer Vision - ECCV 2006*, ser. Lecture Notes in Computer Science, A. Leonardis, H. Bischof, A. Pinz, Eds., Springer Berlin Heidelberg, 2006, vol. 3952, pp. 428–441. Cited on page 128.

[Dragone 11] M. Dragone, S. Abdel-Naby, D. Swords, G. M. P. O'Hare, "Robotic Ubiquitous Cognitive Networks", *ERCIM News*, no. 87, pp. 42–43, October 2011. Cited on pages 22 and 28.

[Dragone 13] M. Dragone, S. Abdel-Naby, D. Swords, G. M. P. O'Hare, M. Broxvall, "A Programming Framework for Multi-agent Coordination of Robotic Ecologies", in *Programming Multi-Agent Systems*, ser. Lecture Notes in Computer Science, M. Dastani, J. Hübner, B. Logan, Eds., Springer Berlin Heidelberg, 2013, vol. 7837, pp. 72–89. Cited on page 22.

[Dudek 08] G. Dudek, M. Jenkin, "Inertial Sensors, GPS, and Odometry", in *Springer Handbook of Robotics*, B. Siciliano, O. Khatib, Eds., Springer Berlin Heidelberg, 2008, ch. 20, pp. 477–490. Cited on page 12.

[Elfes 87] A. Elfes, "Sonar-Based Real-World Mapping and Navigation", in *IEEE Journal of Robotics and Automation*, vol. RA-3, no. 3. 1987, pp. pp. 249–265. Cited on page 50.

[Elfes 89] A. Elfes, "Using occupancy grids for mobile robot perception and navigation", *Computer*, vol. 22, no. 6, pp. 46–57, June 1989. Cited on page 8.

[Fernández-Caballero 10] A. Fernández-Caballero, J. C. Castillo, J. Martínez-Cantos, R. Martínez-Tomás, "Optical flow or image subtraction in human detection from infrared camera on mobile robot", *Robotics and Autonomous Systems*, vol. 58, no. 12, pp. 1273 – 1281, 2010. Intelligent Robotics and Neuroscience. Cited on page 128.

[Feron 08] E. Feron, E. N. Johnson, "Aerial Robotics", in *Springer Handbook of Robotics*, B. Siciliano, O. Khatib, Eds., Springer Berlin Heidelberg, 2008, ch. 44, pp. 1009–1029. Cited on page 7.

[Foka 02] A. F. Foka, P. E. Trahanias, "Predictive autonomous robot navigation", in *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, vol. 1. 2002, pp. 490–495. Cited on page 39.

[Foka 10] A. F. Foka, P. E. Trahanias, "Probabilistic Autonomous Robot Navigation in Dynamic Environments with Human Motion Prediction", *International Journal of Social Robotics*, vol. 2, no. 1, pp. 79–94, 2010. Cited on page 112.

[Foley 97] J. D. Foley, Ed., *Computer graphics*, 2. ed. in C, repr. with corr. ed., Addison-Wesley, 1997. Cited on page 41.

[Fox 03] D. Fox, J. Hightower, L. Liao, D. Schulz, G. Borriello, "Bayesian Filters for Location Estimation", *IEEE Pervasive Computing*, 2003. Cited on pages 32, 33, and 50.

[Fox 99a] D. Fox, W. Burgard, S. Thrun, "Markov Localization for Mobile Robots in Dynamic Environments", *Journal of Artificial Intelligence Research*, vol. 11, pp. 391–427, 1999. Cited on pages 11 and 12.

[Fox 99b] D. Fox, W. Burgard, F. Dellaert, S. Thrun, "Monte carlo localization: Efficient position estimation for mobile robots", *AAAI/IAAI*, vol. 1999, pp. 343–349, 1999. Cited on page 34.

[Gal 00] C. L. Gal, J. Martin, G. Durand, "Smart Office: An Intelligent and Interactive Environment", in *Managing Interactions in Smart Environments*, P. Nixon, G. Lacey, S. Dobson, Eds., Springer London, 2000, pp. 104–113. Cited on page 15.

[Hägele 08] M. Hägele, K. Nilsson, J. N. Pires, "Industrial Robotics", in *Springer Handbook of Robotics*, B. Siciliano, O. Khatib, Eds., Springer Berlin Heidelberg, 2008, ch. 42, pp. 963–986. Cited on page 1.

[Han 12] J. Han, E. J. Pauwels, P. M. de Zeeuw, P. H. N. de With, "Employing a RGB-D sensor for real-time tracking of humans across multiple re-entries in a smart environment", *Consumer Electronics, IEEE Transactions on*, vol. 58, no. 2, pp. 255–263, May 2012. Cited on page 75.

[Hansson 14] S. O. Hansson, "Risk", in *The Stanford Encyclopedia of Philosophy*, Spring 2014 ed., E. N. Zalta, Ed., 2014. Cited on page 100.

[Harper 03] R. Harper, "Inside the Smart Home: Ideas, Possibilities and Methods", in *Inside the smart home*, R. Harper, Ed., Springer, 2003, pp. 1–13. Cited on page 15.

[Helal 09] S. Helal, C. Chen, "The Gator Tech Smart House: Enabling Technologies and Lessons Learned", in *Proceedings of the 3rd International Convention on Rehabilitation Engineering & Assistive Technology*, ser. i-CREATe '09. New York, NY, USA: ACM, 2009, pp. 13:1–13:4. Cited on page 15.

[Hendrich 14] N. Hendrich, H. Bistry, J. Zhang, "PEIS, MIRA, and ROS: Three frameworks, one service robot – A tale of integration", in *Robotics and Biomimetics (ROBIO), 2014 IEEE International Conference on*. Dec 2014, pp. 1749–1756. Cited on page 24.

[Honkavirta 09] V. Honkavirta, T. Perala, S. Ali-Loytty, R. Piche, "A comparative survey of WLAN location fingerprinting methods", in *Positioning, Navigation and Communication, 2009. WPNC 2009. 6th Workshop on*. March 2009, pp. 243 –251. Cited on page 122.

[Isard 98] M. Isard, A. Blake, "CONDENSATION - conditional density propagation for visual tracking", *International Journal Computer Vision*, vol. 28, no. 1, pp. 5–28, August 1998. Cited on page 34.

[Kajita 08] S. Kajita, B. Espiau, "Legged Robots", in *Springer Handbook of Robotics*, B. Siciliano, O. Khatib, Eds., Springer Berlin Heidelberg, 2008, ch. 16, pp. 361–389. Cited on page 7.

[Kidd 99] C. D. Kidd, R. Orr, G. D. Abowd, C. G. Atkeson, I. A. Essa, B. MacIntyre, E. Mynatt, T. E. Starner, W. Newstetter, "The Aware Home: A Living Laboratory for Ubiquitous Computing Research", in *Cooperative Buildings. Integrating Information, Organizations, and Architecture*, ser. Lecture Notes in Computer Science, N. A. Streitz, J. Siegel, V. Hartkopf, S. Konomi, Eds., Springer Berlin Heidelberg, 1999, vol. 1670, pp. 191–198. Cited on page 15.

[Kim 04] J.-H. Kim, Y.-D. Kim, K.-H. Lee, "The Third Generation of Robotics: Ubiquitous robot", in *Proceedings of the 2nd International Conference on Autonomous Robots and Agents*. Palmerston North, New Zealand, December 13–15 2004. Cited on page 26.

[Kim 07] J.-H. Kim, K.-H. Lee, Y.-D. Kim, N. S. Kuppuswamy, J. Jo, "Ubiquitous Robot: A New Paradigm for Integrated Services", in *Robotics and Automation, 2007 IEEE International Conference on*. April 2007, pp. 2853–2858. Cited on page 26.

[Koch 08] J. Koch, C. Armbrust, K. Berns, "Small Service Robots for Assisted Living Environments", in *VDI/VDE Fachtagung Robotik*. Munich, Germany, June 11–12 2008. Cited on page 147.

[Korkalainen 09] M. Korkalainen, M. Sallinen, N. Karkkainen, P. Tukeva, "Survey of Wireless Sensor Networks Simulation Tools for Demanding Applications", in *Networking and Services, 2009. ICNS '09. Fifth International Conference on*. April 2009, pp. 102–106. Cited on page 160.

[Liao 03] L. Liao, D. Fox, J. Hightower, H. Kautz, D. Schulz, "Voronoi Tracking: Location Estimation Using Sparse and Noisy Sensor Data", in *Intelligent Robots*

and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on, vol. 1. October 2003, pp. 723 – 728. Cited on page 39.

[Lucke 08] D. Lucke, C. Constantinescu, E. Westkämper, "Smart Factory - A Step towards the Next Generation of Manufacturing", in *Manufacturing Systems and Technologies for the New Frontier*, M. Mitsuishi, K. Ueda, F. Kimura, Eds., Springer London, 2008, pp. 115–118. Cited on page 15.

[Lundh 06] R. Lundh, L. Karlsson, A. Saffiotti, "Plan-Based Configuration of a Group of Robots", in *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI)*. Riva del Garda, Italy, 2006. Cited on page 20.

[Mao 07] G. Mao, B. Fidan, B. D. O. Anderson, "Wireless sensor network localization techniques", *Computer Networks*, vol. 51, no. 10, pp. 2529 – 2553, 2007. Cited on page 122.

[Martin 14] P. Martin, B.-J. Ho, N. Grupen, S. M. noz, M. Srivastava, "An iBeacon Primer for Indoor Localization: Demo Abstract", in *Proceedings of the 1st ACM Conference on Embedded Systems for Energy-Efficient Buildings*, ser. BuildSys '14. New York, NY, USA: ACM, 2014, pp. 190–191. Cited on page 11.

[Mastrogiovanni 10] F. Mastrogiovanni, A. Sgorbissa, R. Zaccaria, "From Autonomous Robots to Artificial Ecosystems", in *Handbook of Ambient Intelligence and Smart Environments*, H. Nakashima, H. Aghajan, J. Augusto, Eds., Springer US, 2010, pp. 635–668. Cited on pages 25 and 26.

[Meixner 10] G. Meixner, N. Petersen, H. Koessling, "User Interaction Evolution in the SmartFactoryKL", in *Proceedings of the 24th BCS Interaction Specialist Group Conference*, ser. BCS '10. Swinton, UK, UK: British Computer Society, 2010, pp. 211–220. Cited on pages 11 and 15.

[Meyer 08] J.-A. Meyer, A. Guillot, "Biologically Inspired Robots", in *Springer Handbook of Robotics*, B. Siciliano, O. Khatib, Eds., Springer Berlin Heidelberg, 2008, ch. 60, pp. 1395–1422. Cited on page 7.

[Mohan 08] P. Mohan, V. N. Padmanabhan, R. Ramjee, "Nericell: Rich Monitoring of Road and Traffic Conditions Using Mobile Smartphones", in *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*, ser. SenSys '08. New York, NY, USA: ACM, 2008, pp. 323–336. Cited on page 16.

[Montemerlo 02] M. Montemerlo, S. Thrun, W. Whittaker, "Conditional Particle Filters for Simultaneous Mobile Robot Localization and People-Tracking", in *IEEE International Conference on Robotics and Automation, 2002. Proceedings. ICRA '02*, vol. 1. 2002, pp. 695 – 701. Cited on page 47.

[Müllner 11] R. Müllner, A. Riener, "An energy efficient pedestrian aware Smart Street Lighting system", *International Journal of Pervasive Computing and Communications*, vol. 7, no. 2, pp. 147–161, 2011. Cited on page 16.

[Murphy 08] R. R. Murphy, S. Tadokoro, D. Nardi, A. Jacoff, P. Fiorini, H. Choset, A. M. Erkmen, "Search and Rescue Robotics", in *Springer Handbook of Robotics*,

B. Siciliano, O. Khatib, Eds., Springer Berlin Heidelberg, 2008, ch. 50, pp. 1151–1173. Cited on page 7.

[Murtra 08] A. C. Murtra, J. M. M. Tur, A. Sanfeliu, "Action evaluation for mobile robot global localization in cooperative environments", *Robotics and Autonomous Systems*, vol. 56, no. 10, pp. 807 – 818, 2008. Network Robot Systems. Cited on page 25.

[Parsons 92] J. D. Parsons, *The mobile radio propagation channel*, London: Pentech, 1992. Cited on pages 120 and 121.

[Prassler 08] E. Prassler, K. Kosuge, "Domestic Robotics", in *Springer Handbook of Robotics*, B. Siciliano, O. Khatib, Eds., Springer Berlin Heidelberg, 2008, ch. 54, pp. 1253–1281. Cited on page 15.

[Priyantha 00] N. B. Priyantha, A. Chakraborty, H. Balakrishnan, "The Cricket Location-support System", in *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '00. New York, NY, USA: ACM, 2000, pp. 32–43. Cited on page 17.

[Proetzsch 10] M. Proetzsch, *Development Process for Complex Behavior-Based Robot Control Systems*, ser. RRLab Dissertations, Verlag Dr. Hut, 2010. ISBN: 978-3-86853-626-3. Cited on page 86.

[Pu 13] Q. Pu, S. Gupta, S. Gollakota, S. Patel, "Whole-home Gesture Recognition Using Wireless Signals", in *Proceedings of the 19th Annual International Conference on Mobile Computing & Networking*, ser. MobiCom '13. New York, NY, USA: ACM, 2013, pp. 27–38. Cited on page 17.

[Quinlan 93] S. Quinlan, O. Khatib, "Elastic bands: Connecting path planning and control", in *Proceedings of IEEE Int. Conference on Robotics and Automation*. Atlanta, 1993, pp. 802–807. Cited on page 14.

[Reichardt 13a] M. Reichardt, T. Föhst, K. Berns, "Design Principles in Robot Control Frameworks", in *Informatik 2013*, ser. Lecture Notes in Informatics (LNI), M. Horbach, Ed. Koblenz, Germany, September 16–20 2013, pp. 2765–2779. Cited on page 151.

[Reichardt 13b] M. Reichardt, T. Föhst, K. Berns, "On Software Quality-motivated Design of a Real-time Framework for Complex Robot Control Systems", in *Proceedings of the 7th International Workshop on Software Quality and Maintainability (SQM), in conjunction with the 17th European Conference on Software Maintenance and Reengineering (CSMR)*. Genoa, Italy, March 5 2013. Cited on page 151.

[Reichardt 14] M. Reichardt, G. Zolynski, M. Arndt, K. Berns, "On the Benefits of Component-Defined Real-Time Visualization of Robotics Software", in *4th International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR 2014)*, ser. Lecture Notes in Artificial Intelligence (LNAI), D. Brugali, J. F. Broenink, T. Kroeger, B. A. MacDonald, Eds., vol. 8810. Bergamo, Italy: Springer, October 20–23 2014, pp. 376–387. Cited on page 151.

[Reid 79] D. Reid, "An Algorithm for Tracking Multiple Targets", *Automatic Control, IEEE Transactions on*, vol. 24, no. 6, pp. 843–854, December 1979. Cited on page 75.

[Rocco 14] M. Rocco, F. Pecora, S. Sathyakeerthy, J. Grosinger, A. Saffiotti, M. Bonaccorsi, R. Limosani, A. Manzi, F. Cavallo, P. Dario et al, "A planner for ambient assisted living: From high-level reasoning to low-level robot execution and back", in *AAAI Spring Symposium "Qualitative Representations for Robots"*, nessuno. 2014. Cited on pages 23 and 24.

[Ryu 07] H. R. Ryu, M. Huber, "A Particle Filter Approach for Multi-Target Tracking", in *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*. November 2 2007, pp. 2753–2760. Cited on page 32.

[Saffiotti 05] A. Saffiotti, M. Broxvall, "PEIS Ecologies: Ambient Intelligence meets Autonomous Robotics", in *Proc of the Int Conf on Smart Objects and Ambient Intelligence (sOc-EUSAI)*. Grenoble, France, 2005, pp. 275–280. Cited on pages 19 and 20.

[Saffiotti 06] A. Saffiotti, M. Broxvall, B. S. Seo, Y. J. Cho, "Steps toward an ecology of physically embedded intelligent systems", in *Proceedings of the 3rd International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*. Seoul, Korea, 2006. Cited on pages 19, 20, 27, 28, and 29.

[Saffiotti 08] A. Saffiotti, M. Broxvall, M. Gritti, K. LeBlanc, R. Lundh, J. Rashid, B. S. Seo, Y. J. Cho, "The PEIS-Ecology Project: Vision and Results", in *Proc of the IEEE/RSJ Int Conf on Intelligent Robots and Systems (IROS)*. Nice, France, September 2008, pp. 2329–2335. Cited on pages 19, 21, and 29.

[Sanfeliu 06] A. Sanfeliu, J. Andrade-Cetto, "Ubiquitous Networking Robotics in Urban Settings", in *Proceedings of the IEEE/RSJ IROS Workshop on Network Robot Systems*. Beijing, October 10–13 2006, pp. 14 – 18. Cited on pages 24, 25, and 27.

[Sanfeliu 10] A. Sanfeliu, M. R. Llácer, M. D. Gramunt, A. Punsola, Y. Yoshimura, "Influence of the privacy issue in the Deployment and Design of Networking Robots in European Urban Areas", *Advanced Robotics*, vol. 24, no. 13, pp. 1873–1899, 2010. Cited on pages 25 and 29.

[Särkkä 07] S. Särkkä, A. Vehtari, J. Lampinen, "Rao-Blackwellized particle filter for multiple target tracking", *Information Fusion*, vol. 8, no. 1, pp. 2 – 15, 2007. Special Issue on the Seventh International Conference on Information Fusion-Part II Seventh International Conference on Information Fusion. Cited on pages 34 and 74.

[Sathyakeerthy 13] S. Sathyakeerthy, M. D. Rocco, F. Pecora, A. Saffiotti, "Scaling Up Ubiquitous Robotic Systems from Home to Town (and Beyond)", in *Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication*, ser. UbiComp '13 Adjunct. New York, NY, USA: ACM, 2013, pp. 107–110. Cited on page 24.

[Sato 96] T. Sato, Y. Nishida, H. Mizoguchi, "Robotic room: Symbiosis with human through behavior media", *Robotics and Autonomous Systems*, vol. 18, no. 1–2, pp. 185–194, 1996. Cited on page 15.

[Schindler 06] G. Schindler, C. Metzger, T. Starner, "A Wearable Interface for Topological Mapping and Localization in Indoor Environments", in *Location- and Context-Awareness*, ser. Lecture Notes in Computer Science, M. Hazas, J. Krumm, T. Strang, Eds., Springer Berlin Heidelberg, 2006, vol. 3987, pp. 64–73. Cited on page 39.

[Schuetz 14] S. Schuetz, M. Reichardt, M. Arndt, K. Berns, "Seamless Extension of a Robot Control Framework to Bare Metal Embedded Nodes", in *Informatik 2014*, ser. Lecture Notes in Informatics (LNI), E. Ploedereder, L. Grunske, E. Schneider, D. Ull, Eds. Stuttgart, Germany, September 22–26 2014, pp. 1307–1318. Cited on page 151.

[Schulz 03] D. Schulz, D. Fox, J. Hightower, "People tracking with Anonymous and ID-sensors using Rao-Blackwellised Particle Filters", in *Proceedings of the 18th International Joint Conference on Artificial intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003, pp. 921–926. Cited on pages 39 and 74.

[Seo 06] B. S. Seo, M. Broxvall, M. Gritti, A. Saffiotti, J. B. Kim, "Using Javaspace for a PEIS Ecology", in *Proceedings of the 9th International Conference on Intelligent Autonomous Systems (IAS)*. Tokyo, Japan, 2006. Cited on page 20.

[Shin 10] J. Shin, D. Han, "Multi-classifier for WLAN fingerprint-based positioning system", in *Proceedings of the World Congress on Engineering*, vol. 1. 2010. Cited on page 122.

[Sidenbladh 03] H. Sidenbladh, "Multi-target particle filtering for the Probability Hypothesis Density", in *Proceedings of the International Conference on Information Fusion*. Cairns, Australia, 2003, pp. 800–806. Cited on pages 34, 74, 76, and 77.

[Singh 00] S. Singh, R. Simmons, T. Smith, A. Stentz, V. Verma, A. Yahja, K. Schwehr, "Recent Progress in Local and Global Traversability for Planetary Rovers", in *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 2. 2000, pp. 1194–1200. Cited on page 14.

[Sisbot 05] E. A. Sisbot, R. Alami, T. Simeon, K. Dautenhahn, M. Walters, S. Woods, "Navigation in the presence of humans", in *Humanoid Robots, 2005 5th IEEE-RAS International Conference on*. 2005, pp. 181–188. Cited on page 99.

[Sisbot 07] E. A. Sisbot, L. F. Marin-Urias, R. Alami, T. Simeon, "A Human Aware Mobile Robot Motion Planner", *Robotics, IEEE Transactions on*, vol. 23, no. 5, pp. 874–883, 2007. Cited on page 99.

[Spinello 11] L. Spinello, K. O. Arras, "People detection in RGB-D data", in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. Sept 2011, pp. 3838–3843. Cited on page 128.

[Stajano 10] F. Stajano, "Security Issues in Ubiquitous Computing", in *Handbook of Ambient Intelligence and Smart Environments*, H. Nakashima, H. Aghajan, J. Augusto, Eds., Springer US, 2010, pp. 281–314. Cited on page 18.

[Streitz 07] N. Streitz, A. Kameas, I. Mavrommati, Eds., *The disappearing computer: interaction design, system infrastructures and applications for smart environments*, Springer-Verlag, 2007. Cited on page 20.

[Thrun 00] S. Thrun, "Probabilistic Algorithms in Robotics", *AI Magazine*, vol. 21, no. 4, pp. 93–109, 2000. Cited on page 50.

[Thrun 02] S. Thrun, "Particle Filters in Robotics", in *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, ser. UAI'02. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002, pp. 511–518. Cited on page 33.

[Thrun 05] S. Thrun, W. Burgard, D. Fox, *Probabilistic Robotics*, ser. Intelligent Robotics and Autonomous Agents, The MIT Press, September 2005. Cited on pages 8, 9, 12, 34, 35, 36, and 82.

[Thrun 08] S. Thrun, J. J. Leonard, "Simultaneous Localization and Mapping", in *Springer Handbook of Robotics*, B. Siciliano, O. Khatib, Eds., Springer Berlin Heidelberg, 2008, ch. 37, pp. 871–889. Cited on page 34.

[Vo 03] B.-N. Vo, S. Singh, "Sequential Monte Carlo Implementation of the PHD Filter for Multi-target Tracking", in *In Proceedings of the Sixth International Conference on Information Fusion*. 2003, pp. 792–799. Cited on pages 34, 74, and 76.

[Want 92] R. Want, A. Hopper, V. F. ao, J. Gibbons, "The Active Badge Location System", *ACM Transactions on Information Systems*, vol. 10, no. 1, pp. 91–102, January 1992. Cited on pages 16 and 17.

[Weisstein 02] E. W. Weisstein, *CRC Concise Encyclopedia of Mathematics, Second Edition*, CRC Press, 2002. Cited on page 169.

[Wettach 10] J. Wettach, D. Schmidt, K. Berns, "Simulating Vehicle Kinematics with SimVis3D and Newton", in *2nd International Conference on Simulation, Modeling and Programming for Autonomous Robots*. Darmstadt, Germany, November 15–18 2010. Cited on page 158.

[Wichert 12] R. Wichert, "Preface", in *Ambient Assisted Living*, R. Wichert, B. Eberhardt, Eds., Springer, 2012, pp. V – VI. Cited on page 16.

[Widyawan 07] Widyawan, M. Klepal, D. Pesch, "A Bayesian Approach for RF-Based Indoor Localisation", in *Wireless Communication Systems, 2007. ISWCS 2007. 4th International Symposium on*. oct. 2007, pp. 133 –137. Cited on page 122.

[Wille 10] S. Wille, N. Wehn, I. Martinovic, S. Kunz, P. Göhner, "AmICA - Design and implementation of a flexible, compact and low-power node platform", University of Kaiserslautern, Tech. Rep., October 2010. `http://nbn-resolving.de/urn/resolver.pl?urn:nbn:de:hbz:386-kluedo-28076`. Cited on page 145.

[Wille 12] S. Wille, I. Shcherbakov, L. de Souza, N. Wehn, "TinySEP - A Tiny Platform for Ambient Assisted Living", in *Ambient Assisted Living*, ser. Advanced Technologies and Societal Change, R. Wichert, B. Eberhardt, Eds., Springer Berlin Heidelberg, 2012, pp. 229–243. Cited on page 145.

[Williston 09] K. Williston, *Digital Signal Processing: World Class Designs: World Class Designs*, ser. World Class Designs, Elsevier Science, 2009. Cited on page 163.

[Wright 08] C. V. Wright, L. Ballard, S. E. Coull, F. Monrose, G. M. Masson, "Spot Me if You Can: Uncovering Spoken Phrases in Encrypted VoIP Conversations", in *Security and Privacy, 2008. SP 2008. IEEE Symposium on*. May 2008, pp. 35–49. Cited on page 147.

[Youssef 07] M. Youssef, M. Mah, A. Agrawala, "Challenges: Device-free Passive Localization for Wireless Environments", in *Proceedings of the 13th Annual ACM International Conference on Mobile Computing and Networking*, ser. MobiCom '07. New York, NY, USA: ACM, 2007, pp. 222–229. Cited on pages 16 and 17.

[Zuehlke 08] D. Zuehlke, "SmartFactory – from Vision to Reality in Factory Technologies", in *Proceedings of the 17th IFAC World Congress*, M. J. Chung, P. Misra, Eds. 2008, pp. 82–89. Cited on page 15.

Index

AAL, *see* Ambient Assisted Living
Abbreviated Injury Scale, 100
Active Badge Location System, 16
Active Localization, 16
Actuation Model, *see* Motion Model
AIS, *see* Abbreviated Injury Scale
Ambient Assisted Living, **16**, 22, 145, 151
Ambient Intelligence, **15**
Ambient Technology, 1, 15
AnalyzePath, 114
Anonymous Sensor, **28**, 74
ApplyMotion, 48
ApplyToEdge, 105
Arithmetic Mean
    Incremental Update, 168
Artificial Ecosystem, 26
Automated Guided Vehicle, 1
Autonomous Car, 1
Availability, 18
Aware Environment, 1, **15**
Aware Home, 15
Aware Technology, 31

Badge, 16
Bayes' Rule, **33**, 33
Bayesian Filter, 9, **34**, 67, 113
Belief, 13, **34**
BLE, *see* Bluetooth Low Energy
Bluetooth Low Energy, 11
Brownian Motion, 46
Bumper, 149, 150, 159

CAN, *see* Controller Area Network
Candidate Particle, *see* Virtual Obstacle-Avoidance
CDM, *see* Code Division Multiplexing
Code Division Multiplexing, 160
Confidence, 57
Confidentiality, 18
Controller Area Network, 149
Convolution, 104
Coordinate System, 10
    Global, 51
    Sensor, 51

Data Association, *see* Target Tracking
DBN, *see* Dynamic Bayes Network
Dead Reckoning, **11**, 152
Device-Free Localization, 16, 17
Diffraction, 121
Digital Signal Processor, 149
Dirac Delta Function, 104
Discrete Bayes Filter, 35
DSP, *see* Digital Signal Processor
Dynamic Bayes Network, 34
Dynamics, *see* Motion Model

EKF, *see* Extended Kalman Filter
Euclidean Norm, 102
Existing Infrastructure, 4
Extended Kalman Filter, 36

False Negative, 63
False Positive, 63
Fault tolerance, 4
FINROC, 151
Free-Space Path Loss, 120

Galileo, 11
General-Purpose Input/Output, 149
Global Localization, *see* Localization

Global Navigation Satellite System, **11**, 122
Global Path Planning, *see* Path Planning
Global Positioning System, 11
GLONASS, 11
GNSS, *see* Global Navigation Satellite System
GPIO, *see* General-Purpose Input/Output
GPS, *see* Global Positioning System
Graceful Degradation, 71, 143
Graph-Based State Space, 38

HAMP, *see* Human Aware Motion Planner
Hidden Markov Model, 34
HMM, *see* Hidden Markov Model
Human Aware Motion Planner, 99

iB2C, *see* Integrated Behaviour-Based Control
iBeacon, 11
IMU, *see* Inertial Measurement Unit
Industrial Robot, 7
Inertial Measurement Unit, 12
Integrated Behaviour-Based Control, 86
Integrity, 18
Internet, 15, 18
IntersectEllipticConeWithRay, 56
IntersectSensorVolumeWithRay, 56
IntersectSphereWithRay, 56
InVolume, 55

Kalman Filter, 32, **35**
Kidnapped Robot Problem, 11, 13

Laser Rangefinder, 2, 9, 50, 74, 129, 149, 150
Local Path Planning, *see* Path Planning
Localization, **10**, 32, 33, 152
  Global, **11**, 12, 119
  in Aware Environments, 118
  Monte Carlo, **12**, 12, 124, 134
  using Received Signal Strength (RSS) Fingerprinting, **120**, 134
  using Triangulation, 120

Mapping, **7**
  Occupancy Graph, 31, 64, **78**
    Definition, 80
    Experiments, 82
    Initialization, 81
    Update, 81
  Occupancy Grid, **8**, 31, 64, 79, 97
  Topological Map, **9**
Markov Localization, *see* Monte Carlo Localization
MCA2, *see* Modular Controller Architecture 2
Measurement Model, *see* Sensor Model
MHT, *see* Multiple Hypothesis Tracker
Mobile Robot, **7**
Mobile Robot Programming Toolkit, 152
Modular Controller Architecture 2, 151
Monte Carlo Algorithm, 37
Monte Carlo Localization, *see* Localization, 152
Motion Detector, 1, 3, 27, 50
Motion Model, 3, **33**, 37
MRPT, *see* Mobile Robot Programming Toolkit
Multi-Target Tracking, *see* Target Tracking
Multimodal Distribution, 13
Multipath Propagation, 121
Multiple Hypothesis Tracker, **75**, 113

Network Robot, **25**

Observation Model, *see* Sensor Model
Obstacle Avoidance, 3, 15
Occupancy Graph Mapping, *see* Mapping
Occupancy Grid, *see* Mapping
Odometry, 11

Particle Filter, 13, **37**, 105

Particle Filter Localization, *see* Monte Carlo Localization
ParticleWithinLocalSensorRange, 92
Passive Infrared Sensor, 3, 29, 50, 129
  Simulation, 160
Passive Localization, 16
Passive Sensor, 4, 28
Path Planning, 3, 10, **14**, 152
  Global, 14
  Local, 14
  Prediction, **111**
    Experiments, 116
  Risk-Reduced, **98**
    Experiments, 107
PEIS, *see* Physically Embedded Intelligent Systems
Perceptual Model, *see* Sensor Model
PHD, *see* Probability Hypothesis Density Filter
Physically Embedded Intelligent Systems, 19
PIR, *see* Passive Infrared Sensor
Pose, 10, 32, 129
Predictive Path Planning, *see* Path Planning
Privacy, 4, 17, **18**, 28
Probabilistic Model, 3
Probability Hypothesis Density Filter, **76**, 107, 113

Radio Map, **123**
Radio-Frequency Identification, 151
Received Signal Strength, 120
  Fingerprinting, **120**, 122, 134
  Indicator with AmICA Nodes, 146
  Indicator with simulated AmICA Nodes, 160
  Profiling, 122
Recursive Bayesian Estimation, 3, 33
Relevant Particle, *see* Virtual Obstacle-Avoidance
Resampling, 38
RFID, *see* Radio-Frequency Identification
RGB-D Camera, 9, 129, 149, 150, 159
Risk, **100**
  Cost Function, **100**
  Fusion with Distance, **106**
Risk-Reduced Path Planning, *see* Path Planning
Robot Operating System, 151
ROBOT-ERA Project, 23
Robotic UBIquitous COgnitive Network, 21
ROS, *see* Robot Operating System
RSS, *see* Received Signal Strength
RUBICON, *see* Robotic UBIquitous COgnitive Network

Safety Cost Graph, 103
Safety, Low-Cost and Performance, 1, **27**, 141
Security, **18**
Sensor Model, 3, 9, 13, **32**, 37, 50, 129
  Direct, 61
  Direct, Multiple Sensors, 63
  Marginalized Direct, 64
  Marginalized Inverse, 64
Sequential Monte Carlo Method, 37
Service Robot, 1
Simple Path, **88**, 113
Simulation, 157
  Passive Infrared Sensor, 160
  Radio Communication, 160
Simultaneous Localization and Mapping, 34
SimVis3D, 158
Single-Target Tracking, *see* Target Tracking
SLAM, *see* Simultaneous Localization and Mapping
Smart Environment, 1, **15**
Smart Factory, 15
Smart Home, 15
Smart Office, 15
Smart Technology, 1
SMC, *see* Sequential Monte Carlo Method

Sparse Sensor Instrumentation, 4, **27**, 74
State Estimation, 3
State Prediction, 112
State Space, 3, **32**
Symbiotic Robotic Systems, 26
System Model, *see* Motion Model

Target Tracking, 32, 46
  Multi-Target, **72**
    Data Association, 72
    Experiments, 77
  Single-Target, **67**
    Experiments, 68
Topological Map, *see* Mapping
Triangulation, 120
True Negative, 63
True Positive, 63
Twist, 89

Ubibot, 26
Ubiquitous Networking Robotics in Urban Settings, 24
Ubiquitous Robotics, **25**
Ubisense, 11
Ultra-Wideband, 11
Ultrasonic Sensor, 50, 149, 159
Undirected Graph, 9
URUS, *see* Ubiquitous Networking Robotics in Urban Settings
UWB, *see* Ultra-Wideband

Virtual Obstacle-Avoidance, **85**
  Candidate Particle, 87
  Experiments, 93
  Relevant Particle, 87

Wake-Up Robot Problem, **11**, 13, 119
Wheel Encoder, 11, 149, 159
Wheel Odometry, 11, 34
WiFi, 13, 17, 120, 122, 128, 148
Wireless Sensor Node, 3, 11, 31, 68

About this Book

This dissertation shows an approach to optimize the three opposing goals of *safety*, *performance* as well as *cost* of mobile robots that are embedded in aware environments.

It describes how sensor data from aware technology can be used to estimate where people are currently situated in the environment. This is achieved by the use of probabilistic methods for estimating the state of single locations in the environment as well as for tracking of people (in the single and multi-target case). The estimated state is used as an input to the local and global path planners of a mobile robot, enabling safe, cost-efficient and performant mobile robot navigation during local obstacle avoidance as well as on a global scale, when planning paths between different locations.

All the presented methods of this work have been validated in simulation as well as in reality experiments. Even though simulation is an invaluable tool in robotics research, great care has been taken that all the simulation results are also applicable to the real world.

About the Author

Michael Arndt studied computer science from 2006 till 2011 at the University of Kaiserslautern where he attained both his Bachelor's as well as his Master's degree. From 2012 till 2015, as a member of the PhD program of the department of computer science, he was able to conduct his research regarding mobile robots in aware environment at the Robotics Research Lab, the research group led by Prof. Dr. Karsten Berns.